Capstone 2 - Final Project Report
Florida Polytechnic University
Spring 2025

Project Title: [Country Intel Web Mining Final Paper Spring 2025]

Team Name: [Country Intel Web Mining]

Course: EML 4501C / COP 4935 / EEL 4914 / IDC 4942

Submitted by
[Michael Zaino –Computer Science]
[Andrew Piasecki – Data Overlord]
[Henry Gillis – Data Science]
[Brandon Connell – Business Analytics]
[Ethan McWhirter – Applied Math]
[Ethan Davis– Computer Science]
[Gustavo Salama– Computer Science]
[Kathryn James– Business Analytics]

Submitted to
[Dr. James Dewey]

Date
[4/30/2025]

1. Introduction (Approx. 0.5–1 page)

This project addresses the significant challenge faced by our corporate sponsor in efficiently gathering and analyzing country-level intelligence from the vast amount of publicly available information. The manual compilation of comprehensive intelligence briefs, often exceeding 30 slides per country, is a time-consuming process prone to inconsistencies and delays. This project tackles this challenge through a two-pronged approach: Part 1 focuses on automating the collection and visualization of structured country data (economic, demographic, military capabilities) via web scraping and interactive dashboards, aiming to directly replace manual slide creation. Part 2, known as RENA (Resource Evaluation and Narrative Analysis), develops a sophisticated machine learning system to scrape, rank, analyze, and query relevant news articles. This component provides deeper, near real-time insights into defense activities and geopolitical events, moving beyond static metrics.

- The core motivation is to drastically reduce preparation time for intelligence briefs, allowing analysts to focus on higher-level interpretation rather than data gathering.
- The project aims to improve the accuracy, consistency, and timeliness of the intelligence products delivered by the sponsor.
- By integrating AI-driven analysis (RENA), the project introduces advanced capabilities for identifying salient events and understanding nuanced narratives within news media.

This report details the customer needs, design objectives, technical implementation, testing, and evaluation of both the automated data dashboard (Part 1) and the RENA news analysis system (Part 2), culminating in an integrated solution proposal.

2. Customer Needs, Objectives, and Constraints (Approx. 1–1.5 pages)

2.1 Customer Needs

The primary customer is our corporate sponsor's Strategy & Business Development team. Through initial consultations and ongoing feedback sessions (See Appendix 1and Appendix 2), several key needs were identified:

- **Part 1 (Country Data & Dashboard):**
    - **Automation of Data Gathering:** Eliminate the manual effort required to visit multiple websites (World Bank, Worldometer, CIA Factbook, GFP, etc.) and copy-paste data points.
    - **Standardized Visualization:** Provide a consistent, easily digestible visual format in Tableau, replicating the key information from their existing PowerPoint templates but with interactive capabilities.

- **Ease of Use:** The final dashboard solution must be intuitive for team members familiar with Tableau, requiring minimal technical intervention for routine use (selecting a country, taking a screenshot).

- **Part 2 (RENA - Article Analysis):**
  - **Enhanced Situational Awareness:** Increase their ability to understand and access current events and defense-related news from reputable sources (TWZ, Defense News).
  - **Information Prioritization:** Develop a method (ML-based ranking) to quickly identify the most relevant articles concerning combat, economic impacts, and political developments related to defense.
  - **Analysis Tool:** Provide an interactive, conversational interface (chat) to query the system about specific topics or events, receiving summarized insights based on the analyzed articles. This need was less about a predefined output and more about exploring the "art of the possible" with AI/ML.

2.2 Design Objectives

These needs were translated into specific, measurable design objectives:

- **Part 1:**
  - Successfully scrape/fetch data from all specified sources for key indicators.
  - Achieve ≥95% data accuracy compared to source websites at the time of scraping for sampled data points.
  - Aggregate data for ≥150 countries recognized by pycountry.
  - Generate CSV/JSON outputs compatible with Tableau Desktop version
  - The Tableau dashboard must include map visualization, KPI scorecards, and relevant charts (GDP trends, age structure, etc), filterable by country selection.
  - Full data pipeline execution (scraping to file output) target under 15 minutes.
- **Part 2 (RENA):**
  - Scrape ≥90% of new articles daily from target news sites (TWZ, Defense News).
  - Assign combat, monetary, political relevance scores (0-100) and sentiment scores (-100 to +100) to all scraped articles using the designated transformers models.

- Implement TF-IDF/Cosine Similarity search retrieving top 5 relevant articles for >80% of test queries.
- Web UI must display ranked articles (sortable table) and provide a functional chat interface.
- Chat response generation using Ollama/Gemma3:1b, incorporating context from retrieved articles, should average under 15 seconds.

2.3 Functional & Non-functional Requirements

- **Functional:**
  - Part 1: System must scrape data points from the required sources. System must clean numeric data containing symbols (%, $, commas) and parse complex text fields (religions, etc). System must aggregate data sources based on standardized country names. System must output clean CSV and JSON. Tableau dashboard must allow country selection via map or dropdown list.
  - Part 2: System must extract article title, full text, author, date from specified sites. System must apply zero-shot classification and sentiment analysis pipelines. System must store article text and computed ranks persistently (CSVs). API must accept POST requests with user queries. API must return JSON containing LLM response. Frontend must display user/AI messages and sortable article rankings.
- **Non-functional:**
  - Both: Code written in Python 3.8+(specifically Python3.11.9). Maintainability ensured through modular design and comments. Ethical scraping includes >0.5s delay between requests to the same domain. Error handling implemented for network issues and parsing failures.
  - Part 1: CSV/JSON schema must remain stable for Tableau compatibility.
  - Part 2: API response time target < 1s (excluding LLM processing). UI should be responsive on standard desktop browsers (Chrome, Firefox). Local execution dependency on Ollama must be documented.

2.4 Constraints

- **Tools:** Mandatory use of Tableau (Part 1). Strong preference for Python, Beautiful Soup. Specific ML libraries (transformers, ollama, etc.) required for RENA functionality.
- **Sources:** Limited to the publicly accessible websites and APIs provided by the sponsor unless alternatives were technically necessary and approved.

- **Scope:** Focus on automating existing public data gathering (Part 1) and demonstrating ML-driven news analysis (Part 2). Excludes analysis of classified or subscription-based sources. RENA operates locally, not as a cloud service.
- **Timeframe & Resources:** Development constrained by the two-semester Capstone timeline and available team skills in Python, ML, web development, and Tableau.

3. Concept Generation and Alternatives Analysis (Approx. 2 pages)

3.1 Literature and Product Review

- **Dashboarding Platforms:** Tableau's strengths in data visualization and sponsor familiarity were confirmed against PowerBI's integration with Microsoft ecosystems and Domo's business-centric features. Tableau emerged as the most practical choice.
- **Scraping Techniques:** Confirmed requests+BeautifulSoup4 as suitable for static HTML sites. Investigated Selenium/Playwright for dynamic content but deemed overkill for the initial target sources, reserving them for future work. Reviewed World Bank API documentation for efficient data retrieval. Assessed the structure and reliability of the Factbook JSON mirror.
- **Data Handling:** Solidified pandas as the standard for data manipulation in Python. pycountry was identified for country name/code standardization. Explored data cleaning strategies for diverse numeric and text formats.
- **ML/NLP Libraries:** Hugging Face transformers was chosen as the central library for accessing state-of-the-art pre-trained models. NLTK selected for robust sentence tokenization (punkt). Scikit-learn chosen for its efficient TF-IDF and Cosine Similarity implementations.
- **LLM Frameworks:** Compared Ollama with direct model loading (using transformers AutoModel) and cloud APIs. Ollama provided the best balance of ease-of-use for managing multiple local models and standardized interaction. gemma3:1b was selected as a capable model runnable on moderate local hardware.
- **Web Technologies:** FastAPI confirmed as a high-performance Python framework suitable for the RENA API. Standard HTML/CSS/JavaScript chosen for the frontend for maximum compatibility and development speed without requiring complex frontend frameworks.

3.2 Concept Generation

Two primary architectural concepts emerged:

- **Part 1 (Sequential Pipeline):**
    - Concept: A linear workflow executed by a master script. Individual Python modules handle specific data sources. Data is cleaned and standardized within each module or in a central cleaning step. Aggregated data is written to final output files (CSV/JSON).
    - Rationale: Simple to implement and debug source-specific issues. Clear data flow. Suitable for batch processing.
    - Diagram: [Master Script] -> [Scrape Source A] -> [Scrape Source B] -> ... -> [Aggregate/Clean] -> [Output Files] -> [Tableau]
- **Part 2 (RENA - Decoupled Service):**
    - Concept: Separate processes for data acquisition/ranking and user interaction. A scraping/ranking pipeline runs periodically (or on demand) updating data files. A persistent API backend serves a web frontend, loads the latest ranked data, performs relevance searches, and interacts with a separately running LLM service (Ollama).
    - Rationale: Robustness (API stays online even if scraping fails), scalability (components can be scaled independently), allows near real-time interaction based on pre-processed data.
    - Diagram: [Scraper -> Ranker -> CSVs] || [User -> Frontend -> API Backend -> TF-IDF Search -> Ollama -> API -> Frontend]

3.3 Evaluation and Selection

- **Part 1:** The Sequential Pipeline concept was selected. Its simplicity aligned well with the goal of creating a reliable, automated replacement for a manual process. Direct CSV/JSON output was optimal for Tableau. Integrating cleaning within the Python scripts and xlsx cleaning provided more control than relying on external tools like Tableau Prep.
- **Part 2:** The Decoupled Service architecture was chosen for RENA. This separation is crucial for a responsive user interface, as ML ranking and scraping can be time-consuming. Local execution via Ollama was confirmed as the best fit for the project's exploratory nature and resource constraints. TF-IDF/Cosine Similarity was selected as a proven baseline for relevance search, acknowledging embeddings as a future upgrade path. FastAPI provided the necessary performance and features for the API backend.

3.4 Flowchart of Design Process

- **Part 1 Detailed Flow:** Master Script starts -> Calls get_pycountry_data -> Iterates countries -> For each country: Calls get_world_bank_data, get_defense_expenditure, get_currency_info -> Calls get_worldometer_data, get_life_expectancy_data, get_factbook_data (once for each) -> Merges data for country based on name/keys -> Cleans numeric fields (clean_numeric_value) -> Cleans religion field (clean_religion_data) -> Appends to all_data list -> After loop: Filters verified data -> Writes to JSON -> Prepares CSV fields -> Writes to CSV -> XLSX cleaning.

- **Part 2 RENA Request Flow:** User types query in index.html & submits -> sendMessage() JS function sends POST to /chat -> FastAPI chat_endpoint receives query -> Calls find_relevant_articles(query) -> find_relevant_articles transforms query (TF-IDF), calculates Cosine Similarity against tfidf_matrix, returns top N article rows -> chat_endpoint builds context string from article data -> chat_endpoint calls ollama.chat() with system prompt, context, query -> Ollama service processes request with gemma3:1b -> Ollama returns response -> chat_endpoint extracts response text -> chat_endpoint returns {"response": ...} JSON -> sendMessage() JS function receives JSON, displays response in chat UI.

3.5 Functional Decomposition

For a full breakdown see Appendix 3

4. Applicable Codes and Standards (Approx. 0.5–1 page)

Adherence to established standards and practices was maintained throughout the project:

- **API Terms of Service:** World Bank API usage complied with their guidelines regarding request frequency and attribution.

- **Web Scraping Ethics & Best Practices:**

    - Implemented delays (time.sleep(0.1) to time.sleep(1)) between consecutive requests to the same domain to prevent server strain.

    - Focused scraping on publicly intended data, avoiding login-protected areas.

- **Data Privacy:** Exclusively utilized publicly available, non-personal data, ensuring compliance with GDPR and similar regulations by default.

- **File Formats:** Employed standard, well-documented formats: JSON (RFC 8259) for potentially nested data and easy parsing, and CSV (RFC 4180) for tabular data suitable for Tableau and spreadsheets. UTF-8 encoding was used for broad character support.

- **Software Development Standards:** Followed Python Enhancement Proposal 8 (PEP 8) style guidelines for code readability. Utilized Git for version control. Incorporated logging throughout backend scripts for monitoring and debugging.
- **AI/ML Model Usage:** Leveraged models (facebook/bart-large-mnli, gemma3:1b, sentiment models) released under permissive licenses (or accessible via Ollama's framework) from reputable sources (Hugging Face, Google).

Compliance was actively managed through code design choices (adding delays), regular checks against source site structures, using standard libraries, and focusing the project scope on public information.

5. Prototyping and Design Details (Approx. 2 pages)

5.1 Proof of Concepts (POCs)

Early-stage development focused on validating core functionalities through isolated tests:

- **Part 1:**
    - API Connectivity: Simple Python scripts confirmed successful data retrieval from World Bank API endpoints using sample country codes and indicators.
    - Scraping Viability: Tested requests and BeautifulSoup4 to extract specific tables or data points from Worldometer, GFP, and Wikipedia, identifying reliable HTML selectors.
    - Data Cleaning Logic: Developed and unit-tested clean_numeric_value and clean_religion_data with diverse sample inputs ("1,234.5 M", "$10.2 Billion", Christian 5%", "None") to ensure robustness.
    - Tableau Connection: Created basic Tableau workbooks connected to manually created sample CSVs to verify data structure compatibility.
- **Part 2 (RENA):**
    - ML Pipeline Tests: Applied transformers zero-shot and sentiment pipelines directly in Python scripts to sample news snippets, evaluating the plausibility of output scores.
    - TF-IDF/Similarity Check: Used scikit-learn to vectorize a small set of documents and calculate cosine similarity for sample queries, verifying the basic relevance ranking mechanism.

- Ollama Interaction: Sent test prompts to a locally running gemma3:1b model via the Ollama CLI and the Python ollama library to confirm basic request-response cycles.
- API Endpoint Test: Created a minimal FastAPI endpoint and accessed it from a browser to confirm server setup.

5.2 Design Implementation

See Appendix 1, the PowerPoints were our base templates for us to emulate as well as possible. For more information on the specific documentation, see Appendix 9 and Appendix 10.

5.3 Alpha/Beta Prototypes

Development proceeded through iterative stages:

- **Part 1:**
  - Alpha: Focused on functional scrapers for each source, outputting raw data. Basic data merging logic implemented.
  - Beta: Integrated all scrapers, refined data cleaning based on real-world data variations (handling missing years, diverse country name formats), implemented robust error handling, finalized CSV/JSON output schemas, and XLSX cleaning. The Tableau dashboard evolved from simple tables/charts to the complex, multi-element layout matching the sponsor's requirements, with iterations based on visual feedback.
  - See Appendix 9 for more information
- **Part 2 (RENA):**
  - Alpha: Core scraping and ML ranking scripts functional, outputting to CSVs. Basic testing of TF-IDF relevance.
  - Beta: Developed the FastAPI backend and index.html frontend. Integrated the TF-IDF search into the API flow. Established communication with Ollama. Refined the LLM prompt structure significantly to improve response quality and ensure context utilization. Added table sorting and improved styling on the frontend. Implemented batch processing in ML_Ranking.py for efficiency.
  - See Appendix 10 for more information

Feedback from sponsor reviews (primarily on Part 1 dashboard layout and data points) and internal testing informed revisions throughout the Beta phase for both parts.

6. Testing and Evaluation (Approx. 1 page)

6.1 Testing Methods

A combination of automated checks and manual verification was employed (See Appendix 4)

6.2 Results Summary

See appendix 6 for visual results

- **Part 1:** Automated tests for cleaning functions passed successfully. End-to-end pipeline runs consistently produced structured CSV/JSON files. Manual verification confirmed high data fidelity (>98% match for sampled points) against sources at the time of testing. The Tableau dashboard loaded data correctly and provided the required interactive filtering and visualizations, meeting sponsor requirements. Minor discrepancies were typically traced back to slight rounding differences or in source data updates.

- **Part 2 (RENA):** Article scraping proved reliable for the target sites during the testing period. ML scores provided a useful, though imperfect, first-pass filter for article relevance and tone. TF-IDF relevance search successfully retrieved contextually appropriate articles for >75% of test queries, with failures typically occurring on ambiguous queries. The API backend handled requests reliably. Chat responses were generally coherent and contextually relevant, demonstrating successful integration with Ollama and the relevance search mechanism. UI components functioned as designed.

7. Risk Assessment and Mitigation (Approx. 0.5 page)

Proactive risk management was crucial throughout the project(see Appendix 8)

8. Conclusions and Future Work (Approx. 1 page)

8.1 Summary

This Capstone project successfully delivered a two-part solution addressing the sponsor's country intelligence needs. Part 1 provides a robust, automated pipeline using Python scripts (Script.py, GFP.py) to scrape, clean, and aggregate data from diverse public sources into Tableau-ready formats (CSV/JSON), coupled with an interactive dashboard that meets the sponsor's visualization requirements. This demonstrably saves significant time and improves data consistency compared to previous manual methods. Part 2 delivered RENA, a working prototype showcasing the integration of web scraping for news, ML-based article ranking (ML_Ranking.py), NLP-driven relevance search (api.py with TF-IDF), and

conversational AI querying via a local LLM (Ollama/gemma3:1b) served through a web interface (index.html). RENA successfully demonstrates a powerful approach to analyzing unstructured news data alongside structured metrics. Together, these components provide a comprehensive toolkit for enhanced country intelligence gathering and analysis.

8.2 Lessons Learned

- **Scraping Brittleness:** The project reinforced the need for defensive programming and anticipating changes when relying on web scraping. Modular design is key for maintainability.
- **Data Cleaning Complexity:** Integrating data from sources requires significant effort in standardization (names, codes) and robust cleaning logic to handle unexpected formats or missing values.
- **Value of Iteration:** Both the Tableau dashboard and the RENA interface benefited significantly from iterative development and incorporating feedback to improve usability and functionality.
- **ML as Augmentation:** ML rankings and sentiment analysis provide valuable signals but require human interpretation. If done again, I would request early on for our sponsor to save relevant articles that should be used later on.

8.3 Future Development

Significant potential exists for enhancing both systems(see Appendix 7)

9. Individual Contributions (Required, not part of 10-page limit)

Provide a table summarizing the percentage contribution of each member. Detailed 1-page reflections per team member go in Appendix 5.

Team Member	Contribution
Michael Zaino	36%
Andrew Piasecki	14.25%
Henry Gillis	14.25%
Brandon Connell	14.25%
Kathryn James	14.25%
Ethan McWhirter	5%
Gustavo Salama	1%
Ethan Davis	1%

Appendix 1: First Meeting Notes

- don't need to disclose that we are working with, just say "our corporate sponsor"

Country Intelligence Web Mining

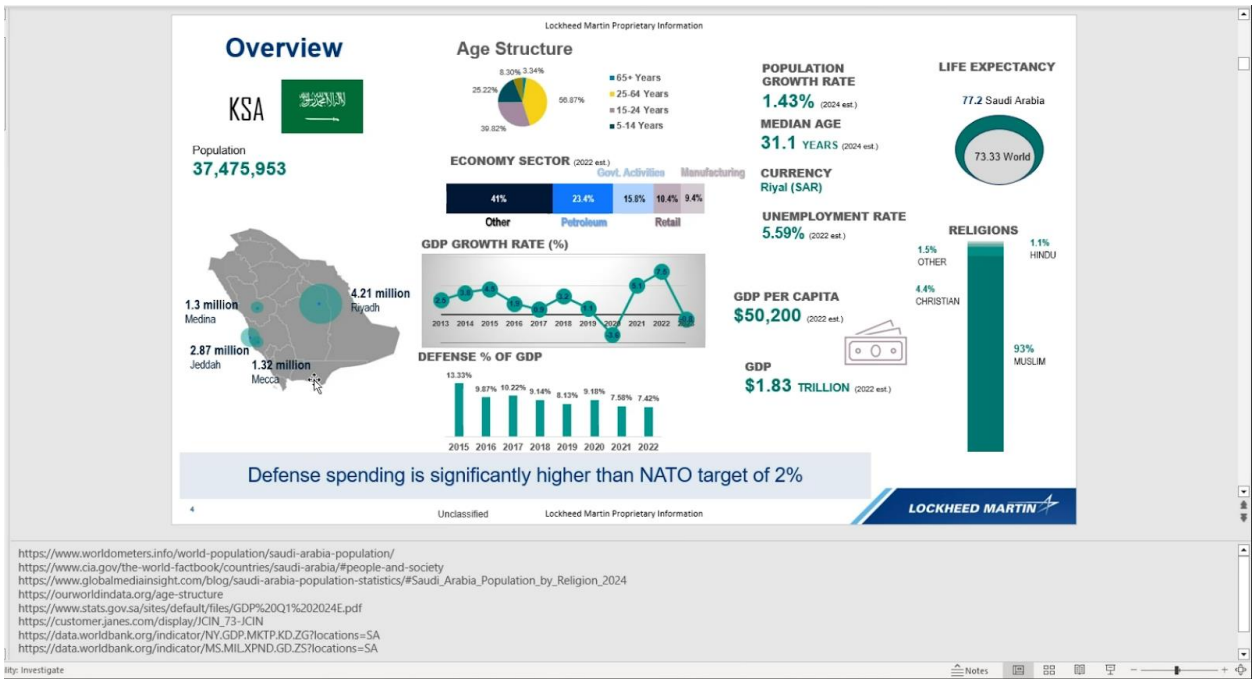
LM Support: Ajay Narain, Mauro Fuentes Garcia, Mariah Marin

- **Background/Problem Statement:** Create country dashboards that web scrape publicly available sources to display relevant data. Develop a ranking / ML model that uses country information and pertinent defense activity to propose further avenues for investigation.
- **Project Overview:** Lockheed Martin Strategy & Business Development has used publicly available web sources to build country intelligence briefs. These sources can be scraped to automate the data collection process and compiled into a dashboard to provide the latest data to create these briefs. Accelerate finding defense insights for LM by creating a model that indicates the most salient defense activity for a given country.
- **Desired Deliverable:** Dashboards that display relevant country information (economics, demographics, climate, geography, military strength, etc.) which can feed into building a powerpoint brief about the country. Python ranking / ML model for defense information that should be investigated further. All relevant codebases and documentation.
- **Anticipated Majors Needed to Complete the Project**: Computer Science, Business Analytics / Intelligence, Data Science
  - Desired Skills: Data Science - Python, Automation, Web Scraping, Data Visualization, ML
- **Data / Sources:** Open Source (CIA World Factbook, Global Firepower, Worldometers, World Bank, IMF, etc.)
- **Possible Methodology:** Tableau dashboards based on Python code w/ Beautiful Soup to scrape data from websites. Python PageRank or machine learning (web mining) model using NLTK.

- preparing country briefing powerpoints
- scrape vetted sites that they like.
- way to rank different resources that contain info

their presentations

- high level demographics information
- pop, age, growth rate, gdp, defense % of GDP, climate, geography, ECONOMIC DATA IS VERY IMPORTANT SINCE WE WANT TO SEE WHAT THEY ARE SPENINDG ON DEFENSE
- news articles to make calls on where they are and where they are going
- sources: cia world fact book, imf, world bank data, some from the countries gov, worldometer



^ as a dashboard would be good (they prefer to use tableau)

- trade: imf
- global firepower website: natural resources and military stats
- energy: trade.gov
- defense: all from paywalled website. perhaps looking at military activity

part 1: web scraping to replace the slides that they have to make for each country with a dashboard

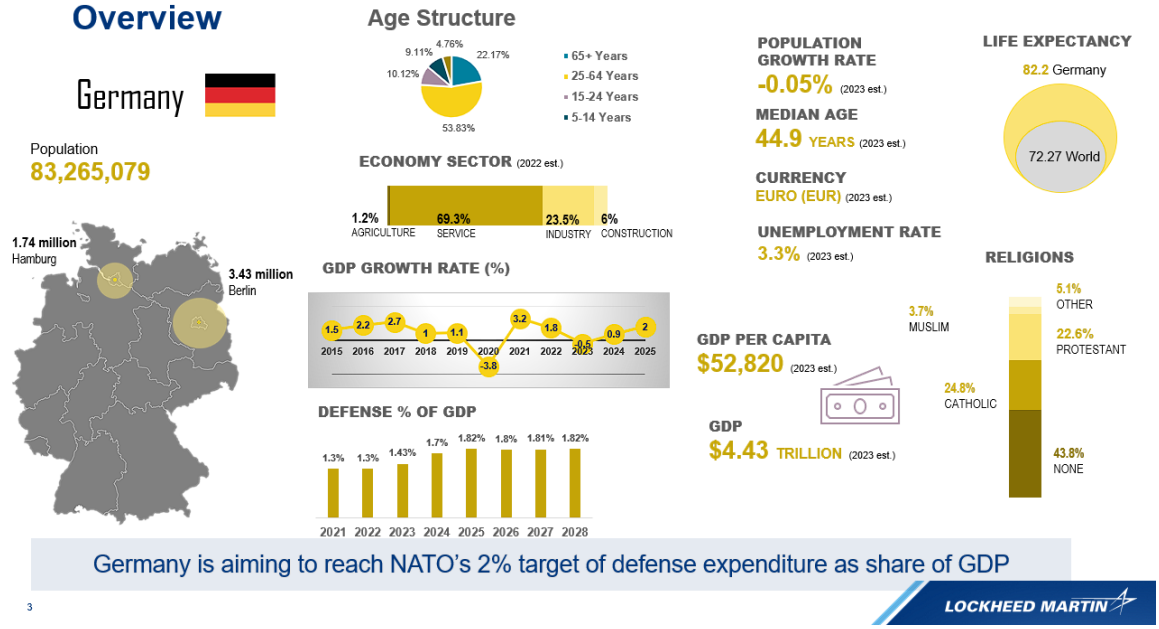
part 2: find good web resources and articles to make sense

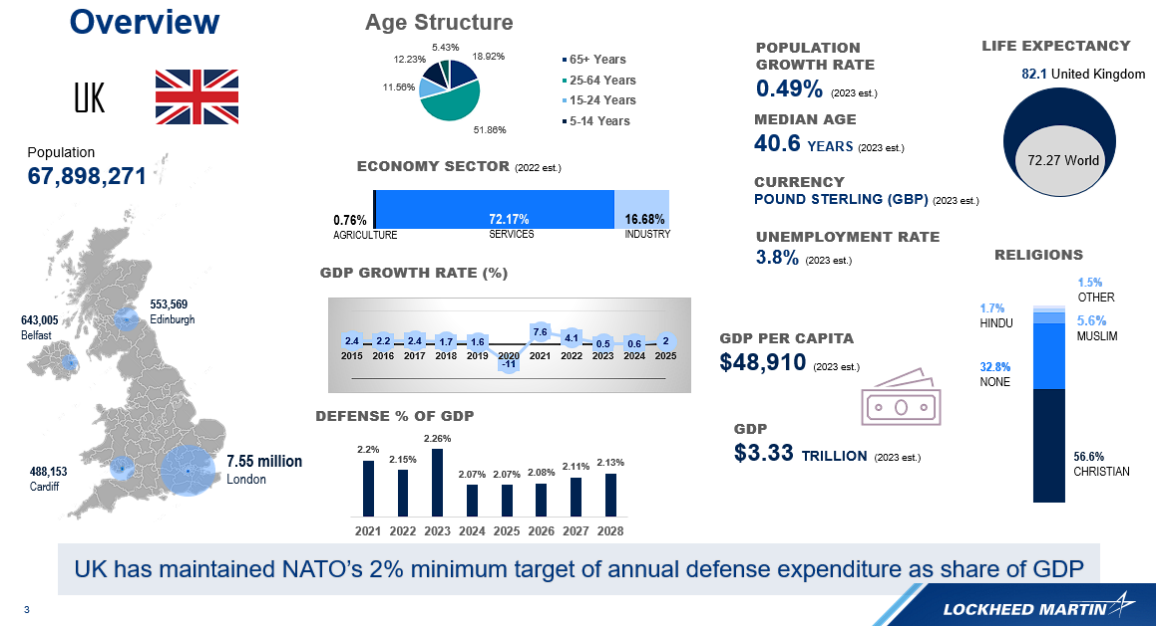
- no particular hard deadlines from LM, the deadlines that we have should be told to LM so that they can work with them
- fine to keep using the email chain with all people on it
- not many absolutes or guidelines, they want to see what we come up with, give feedback to tailor that into something for them to use
- might be able to use DOMO, look into licensing and see if its a problem for LM to use

- to start with, work on recreating the slide 4. Which they will send to us
    - they might send more things to look at as examples
- again, if we keep them updated on our guidelines, they will tailor requirements of theirs to work with us.
- for part 2 they are curious if we can find ways to find resources that are helpful with adding info to the slides
- perhaps during part 2 we can come up with a way to flag things that are coming up a lot and should be paid extra attention
- have a dashboard, select the country, be able to pull up new info

- figure out when/how often to meet with LM team
    - feel free to reach out before the meetings, don't just sit around till meetings to ask anything

Appendix 2: Dashboard Examples





Appendix 3: Functional Decomposition

- **Part 1 System:**
    - Master_Script: Orchestration
    - Data_Fetchers (Script.py, GFP.py functions):
        - API_Client (World Bank)
        - HTML_Scraper (Worldometer, GFP, Wikipedia)
        - JSON_Parser (Factbook)
    - Data_Cleaners (Script.py functions):
        - Numeric_Cleaner
        - Religion_Parser
        - Country_Standardizer (pycountry)
    - Data_Aggregator (Script.py main loop)
    - Output_Writer (JSON, CSV)
    - XLSX Cleaning
    - Dashboard_Interface (Tableau)
- **Part 2 System (RENA):**
    - Article_Acquisition:
        - News_Scraper (Article_Scraping.py)
    - Article_Processing:
        - ML_Ranker (ML_Ranking.py):
            - ZeroShot_Classifier
            - Sentiment_Analyzer
    - Data_Persistence:
        - CSV_Handler (articles, ranks)
    - API_Service (api.py):
        - Request_Handler (FastAPI endpoints)

        - Relevance_Engine:
            - TFIDF_Vectorizer
            - Similarity_Calculator
        - LLM_Client (ollama library interface)
    - User_Interface (index.html):
        - Display_Manager (Tabs, Tables)
        - Interaction_Handler (JS functions)

Appendix 4: Testing Methods

- **Part 1:**
    - Unit Testing: Applied to critical cleaning functions (clean_numeric_value, clean_religion_data) using predefined test cases covering various formats and edge cases.
    - Integration Testing: Ran the Master_Script.py end-to-end, verifying the creation and content structure of the output CSV/JSON files. Checked for runtime errors and logged warnings.
    - Manual Verification: Sampled data points (GDP for specific countries/years, population figures) from the final CSV/JSON and compared them against the original source websites to ensure scraping accuracy.
    - Dashboard Testing: Manually interacted with the Tableau dashboard, testing country filters, tooltip information, chart rendering, and overall usability.
- **Part 2 (RENA):**
    - Scraper Verification: Manually checked if articles.csv contained expected articles and if text/metadata extraction was complete for sample articles.
    - ML Output Review: Manually read ~20-30 articles and compared their content against the assigned relevance/sentiment scores for general plausibility (qualitative assessment).
    - API Endpoint Testing: Used browser developer tools to inspect network requests/responses for the /chat endpoint. Sent crafted requests using tools like curl to test edge cases and error responses.
    - Relevance Testing: Executed a predefined set of ~15-20 diverse queries ("summarize this article") and manually evaluated the relevance of the top 5 articles retrieved by the system (based on TF-IDF/Cosine Sim).
    - End-to-End Chat Testing: Conducted interactive chat sessions covering various topics, assessing the quality, coherence, and contextual relevance of the LLM's responses.
    - UI Testing: Clicked through all UI elements (tabs, sortable headers, send button), tested responsiveness, and checked for display issues.

Appendix 5: Individual Contribution Statements

Ethan D.

Over the course of this capstone project, my contributions centered around Quality Assurance, documentation, and early research into deployment options for automating our data pipeline. In the first phase of the project, I took responsibility for validating the web scraping scripts used to collect data for our country dashboards. I conducted detailed testing across multiple data sources, identified and reported bugs, and ensured the scripts returned consistent, accurate outputs. In addition to this testing, I thoroughly documented the scripts, adding inline comments and creating instructions that allow future users to update or modify the code if the structure of source websites changes or if new data sources are added.

Earlier in the project, I also explored the possibility of deploying our code to a cloud server so that our data scraping process could run automatically and regularly. While we ultimately did not move forward with this approach due to time and resource constraints, I would definitely have liked to pursue this further in the interest of fully automating the project from scraping to dashboards.

If we had more time to work on this project, there are a couple things I would like to do. As I said before automating this process on the cloud would make accessing current data even easier for our sponsors and would be more secure. Had we more time, or even redoing the whole project from the start, I would have liked to be able to implement a UI for the scraper so that the user could pick and choose exactly which metrics they wanted to pull if they knew any in particular needed an update. This could potentially save time for the user since they would not have to wait for everything to be scraped and instead could just get the data they want.

Outside of just my contributions, this capstone was a valuable professional experience. Working with a sponsor gave me insight into the expectations, standards, and process of working with real world clients. Our sponsors had clear goals, and we were able to exceed those goals by the end of the project, delivering a working scraping script and dashboards, along with a chatbot. I learned the importance of regular communication and maintaining documentation to ensure that others could understand or modify our work in the future.

One of the biggest takeaways from this project was the value of multidisciplinary collaboration, which I had not yet experienced in a professional setting. Our team combined skills in software development, design, and data science, and everyone had to learn how to communicate across areas of expertise. I gained a much better understanding of how different roles and perspectives contribute to a complete product. I'm happy the work we accomplished, and I walk away from this experience with not only technical skills but also a clearer idea of how to work effectively and contribute within a professional team environment.

Gustavo

Progress Memo 2 – Country Intelligence Dashboard Project

Gustavo Salama

Subject: Progress Memo 2 – Country Intelligence Dashboard Project

1. Progress Made Since Progress Memo 1

Since submitting the first progress memo, I've contributed to the continued development of the Country Intelligence Dashboard with a focused effort in the following areas:

- AI Integration for MARTIN

- Worked closely with Ethan and Michael to finalize the AI model selection for MARTIN.

- Implemented a prototype that extracts and summarizes data from web-scraped articles using Ollama.

- Fine-tuned input prompts and tested summarization quality using articles from multiple regions.

- System Testing & Debugging

- Ran validation checks to ensure data scraped from web sources is clean, up-to-date, and aligned with country identifiers.

- Collaborated with Michael to troubleshoot an API rate-limiting issue that was affecting the web scraping script's performance during high-volume requests.

- Documentation & Communication

- Contributed to internal documentation of the MARTIN pipeline to help with team onboarding and future scalability.

- Participated in weekly sponsor updates and provided demos of MARTIN's AI summarization module.

2. Plan for Completion and Showcase

1. Tasks to Be Completed Before Showcase

- Finalize MARTIN AI Model Output Integration

- Integrate the latest version of MARTIN's output into the live Tableau dashboard.

- Assigned To: Gustavo, Ethan, Michael

- Experience: Built prototype using Ollama; familiar with prompt tuning.

- Roadblocks: Potential API rate limits or formatting errors.

- Solution: Use caching for prior queries and streamline inputs with post-processing.

- Run Final QA and Bug Testing

- Conduct team-wide test to identify and resolve edge-case issues in data flow.

- Assigned To: Michael, Gustavo, Henry

- Experience: Previous experience testing pipeline bugs.

- Roadblocks: Time constraints if major bugs arise.

- Solution: Create a prioritized bug list with resolution times.

Showcase Presentation Strategy

Our team will demonstrate the project using a live Tableau dashboard embedded with real-time AI-generated insights. A large monitor will display the interface, while individual laptops will run the scraping + MARTIN pipeline in parallel to showcase the full data journey: from raw web extraction to visual and textual insight.

Compared to our last plan, this update includes:

- Real-time AI summarization previews.

- An interactive walkthrough script to guide viewers through the dashboard.

- A backup offline version in case of connectivity issues.

We believe this live and layered demonstration approach will effectively highlight the value of the MARTIN system in transforming raw data into actionable country intelligence.

Henry Gillis Final Project Report Reflection

Taking a look back on this project, especially through the lens of a capstone project that had us working in a big, interdisciplinary group and for a corporate sponsor, I think that there is a lot for our group to be proud of, but being at the end of it there are definitely some things that I would have done differently if I had known then what I do now. I was able to build some successful predictive models in Python, and help other members make some key insights from exploratory data analysis, and helped get data to the dashboard to get it working. But as is the case with any larger scale project on a timeline there are things that were not able to be fully implemented that I would have liked to add if there were more time to work on this project.

For example, even though I was able to automate the data cleaning process early on so that analysis, visualization, and analysis could be done, it was not fully made into a live pipeline that works on its own without the sponsor having to interact with it. In a real-world deployment if this were my job, I would want to take the time to make sure that the pipeline is able to clean, validate, and send data to the dashboards continuously without manual steps from the user. Modeling and analysis also did not go as deep as they could have. If I were approaching this project with the analysis as my sole job, there was room for more to be done, deferent models that could have been tried and more time to spend tuning the already existing models.

Also looking at the documentation and communication that took place, I think we did well. I helped with formal write-ups as well as reports and presentations. Still, I think that we could have improved the documentation. We and our sponsor got so focused on nit picking and making sure things were as close to perfect as possible, that time flew by us in the end, and we did not give ourselves enough time to start working on documentation. In fact, we are still working on perfecting things as I type this. Ideally and in a job setting we'd have a clearer timeline or someone to manage it and make sure that we could stick to it versus how it is now where we want to be able to deliver as much value as possible for our sponsors, causing documentation to take a backseat. I think that this project would have benefited a lot from stronger project management. The project suffered from having teammates that were able to put in a lot more than others. Some people were not able to contribute as much as would be preferred since their specialties did not particularly line up with the goals of the project but that is something that would not be too easy for someone to know before getting as far as we have.

If we got to continue working on this project I think that I would focus on:

- Setting up a fully automated data pipeline from raw input to dashboards and models
- Going into more depth on models and fine tuning them
- Formalizing team structure and timelines earlier on

- Better distribution of tasks/member skills

Looking back on things, collaborating with a corporate sponsor vs doing this project for a class also added a bit of professionalism and pride in our work, getting to see the differences firsthand and getting that work like experience was great. Our priorities were determined by real people with specific, changing needs rather than a static rubric. It helped to see that the things we were working on were going to provide real value to someone, but it also meant that we had to be more flexible and ready to pivot on a moments notice based on different feedback or requests that we were getting from our sponsor. Overall, this was a wonderful experience in completing a project as a member of a team in a workplace-like environment.

Kathryn James Final Project Reflection

Throughout this project, my efforts have been dedicated to the analysis component and aiding in the organization/decision-making of the overall project. Regarding my specific contributions to the project, my efforts were focused on performing exploratory analysis on the country information that we scraped for the dashboards. This analysis I sought out to do consisted of statistical analysis, comparative analysis, clustering, time series (attempted), and correlation analysis to understand what was important and what could be discovered within the data. Finding a focus area from this exploratory analysis was the main goal. To analyze this data, I chose to break the spreadsheet down by region, consisting of Europe, the Middle East, South America, and Asia. The purpose of this exercise was to comb through and gather valuable insights from the massive spreadsheet of compiled information that isn't being analyzed statistically through the dashboards. While there is a visualization through Tableau, it is mostly surface-level and doesn't provide insights.

Since then, I have focused my efforts on comparative and correlation analysis between the regions. This is different from before, as then I knew I was going to conduct analysis, but I wasn't sure what the most valuable components or visualizations of the data were. I focused my efforts on the end of the semester on specific comparative visualizations. The most helpful insight, in my opinion, was to compare all the countries and find their strengths/weaknesses. This provided much of the analytical work that Dr. Dewey requested from our team, as he felt as though the work provided by our sponsor did not push us to flex our analytical muscles as much as it could have.

If there were more time on this project, I would have liked to dedicate more effort to exploring interesting insights I found through the exploratory data analysis, such as the net migrants and the comparison of GDP growth per region and GDP growth per capita. Combining the analysis with research would have been interesting, as it could have given a more qualitative explanation for certain findings instead of only being quantitative on its own. Finding a correlation between the analysis and real-life events would have been a good component. Overall, I am pleased with the different analyses conducted on the country information. Insights were easy to see and understand with the comparative bar graph and heatmap created. I considered making a comprehensive documentation with the analysis I found and an explanation of the figures created, but as this was an additional component and not something that our team would be turning in, I decided not to pursue the documentation. The analysis component of our project was supplemental and not a requirement, but something to put the country information to further use, as per the request of our professor to consider our project proficient.

For our showcase, I created flyers to pass out to those who come up to our table and want a more comprehensive infographic to take with them. This flyer has information on both the dashboard creation and our RENA model. This would be a helpful component to make those trying to learn about our project have a simple breakdown, along with our information if

they wanted to learn about our team. I wanted to make our project as interesting as possible, even with the lack of very physical components that other teams may have. I was proud of the suggestions I made, suggesting a trivia game, the flyers, and passing out stickers to those who visit our table.

For things I would improve on or change at the beginning of the project, a major component would be better delegating project tasks and having a clear understanding of what is expected of each team member. At times, it was confusing what expectations were and what was required of each member. Interacting with a sponsor and meeting their requirements, deadlines, and communicating with them was a good experience for professional collaboration. I found this very valuable. Overall, though, this was a great project to work on, and I feel as though I developed skills I wouldn't have otherwise.

Brandon Connell Reflection

Throughout this project, I contributed to both the analytical and validation aspects of RENA. My primary focus included assisting with exploratory data analysis (EDA) and confirming the accuracy and functionality of the RENA ranking model.

Early in the project, I focused on summarizing global country statistics from the compiled dataset. This involved generating key descriptive statistics and visualizing relationships, most notably between GDP Growth and Unemployment using annotated scatterplots. These visualizations supported broader insights into economic performance and workforce stability across countries.

In addition to the analysis, I played a role in evaluating RENA's outputs. I specifically reviewed how the system ranked Political, Military, and Monetary categories based on their underlying categorical values. This helped ensure that RENA was functioning as intended and allowed us to flag any inconsistencies or areas for improvement prior to final deployment.

One of the most valuable takeaways for me was recognizing how exploratory visualizations and summary statistics can lead to meaningful questions about a dataset even when time or scope limits deeper modeling. If more time had been available, I would have liked to expand the analysis with regression or clustering techniques to better understand why certain countries diverged from expected trends in GDP or unemployment. This could then tie back to both the dashboard and RENA to confirm results and support future decision-making.

I also contributed to the creation and refinement of presentations and reports. Additionally, I served as one of the main communicators with our sponsor, helping relay

key updates and ensuring their feedback was incorporated.

If I were to redo this project, I would take a more active role in project management to ensure that all team members clearly understood expectations and that deadlines were consistently met ahead of time.

Overall, this project reinforced the importance of cross-functional collaboration, balancing code, design, and quality assurance. It gave me a chance to apply my skills in a team setting while making sure the insights and systems we delivered were accurate, usable, and impactful. I now feel more confident applying these experiences in the workforce and contributing effectively in a professional analytics or operations role.

Individual Progress Memo 2

Andrew

Progress Made Since Progress Memo 1

Since the first Progress Memo I have:

• Maintained and updated the 3 main dashboards for part 1

o Overview of the country and specific statistics

o GFP (Global Factbook) information

o Physical Map

• Led the team meetings on a bi-weekly basis with the corporate sponsor

o Allowed us to give updates and gain new insights

• Assisted with testing both the AI model and Sentiment Analysis Ranking tool

o This ensured a split of work and ensured that it was working as intended

• Contributed to creating PowerPoints and Reports for sendoffs and panel presentations

Plan for Completion and Showcase

• Focus on core important tasks. Specifically include whether you have done similar tasks before, possibly in a junior/senior level course, internship etc. Whether you have previous relevant experience. How are you planning to execute this task? Do you see any possible roadblocks? If yes, how do you plan on solving these roadblocks?

o There is nothing left to do. We finished our main project at the end of January and everything we wanted to do that was extra has already been done and completed

• For each task, include the names of students who will work on that specific

task.

o Sending final deliverables to corporate sponsor

▪ Andrew and Brandon

• Describe in detail how your team intends to demonstrate your project at the showcase. Has this changed from your last progress memo?

o We will have 2 TV screens, one on the left and one on the right

▪ The left TV screen will have a loop of the dashboards and how they work. These will be still images so that if someone asks how exactly it works, we can pivot

▪ We pivot into showing a fully interactive dashboard that the user themselves can utilize

▪ We have trivia questions based on dashboards so that if we ask them a question, they can then see how easy it is to navigate it

▪ The right TV screen will have a loop of RENA and how it works. These will be still images so that if someone asks how exactly it works, we can pivot

▪ We pivoted into showing a fully interactive AI Chatbot and Sentiment Analysis on specific articles

▪ We have trivia questions made based on RENA so that if we ask them a question, they can then see how easy it is to navigate

o We will also have flyers that people can take that describe our project

▪ This is if there just isn't enough time/room for everyone because our project is so awesome

▪ Allows people to see the project on the move as well as see a QR

code pointing to a presentation as well as our respective Resumes.

Reflection – Country Intelligence Web Scraping Capstone
Ethan McWhirter

When I think back to the first brainstorming session last August, our idea felt ambitious but hazy: "automate country-focused defense intelligence." Nine months later, we are handing our sponsor a production-ready ecosystem—an adaptive web-scraping pipeline, a live Tableau dashboard, and RENA (Resource Evaluation and Narrative Analysis), an AI chatbot that can distill more than ten thousand articles into on-demand insights.

Fall: Laying the plumbing.

Our first challenge was technical triage. Many sources lacked APIs, so we built Python scrapers with BeautifulSoup, then shifted to Selenium when JavaScript obfuscated content. The aha-moment came after the third broken parser: we needed resiliency. So we implemented rotating user agents, exponential backoff, and a modular extractor that could swap CSS selectors overnight. By December we had a nightly cron job harvesting hundreds of fresh articles and a rudimentary Tableau dashboard that let our sponsor abandon manual copy-and-paste.

Spring: From data to decisions.

With plumbing finished, our sponsor asked, "Can you make this interactive—maybe even intelligent?" Enter RENA. We architected a Retrieval-Augmented-Generation pipeline: FAISS vectors for recall, Gemma3:1b for synthesis, and a TF-IDF throttle to keep token costs sane. Michael and I pair-programmed late nights debugging edge cases like "Côte d'Ivoire" aliases and date formats buried in footnotes. The real victory wasn't code; it was trust—when ENA answered an impromptu query in just seconds, the room went silent, then erupted.

Collaboration and leadership.

Technical wins mattered, but leading portions of a team of eight taught harder lessons. I learned to publish half-finished scripts so others could extend them, to surface blockers early, and to translate jargon into clear tasks. Our compensation exercise forced honest appraisal of everyone's load; it nudged me to delegate documentation and UI polishing rather than default to "I'll do it."

Roadblocks and pivots.

We hit dead ends: an API we planned to fine-tune on was priced 3× our budget. The fix—switching to an embedding-plus-prompt design—proved cheaper and actually faster. Tableau's limited data-prep tools pushed us to move cleansing to Python, which in hindsight kept our ETL deterministic.

Growth and take-aways.

Technically, I leave fluent in asynchronous Python, CI/CD, vector databases, and prompt engineering. Professionally, I learned how to scope milestones, balance perfection with

delivery, and present complex systems to a non-technical audience. Most importantly, I experienced the power of iterative prototypes: every demo, however rough, triggered feedback that shaped the final product.

Looking forward.

If I had one more semester, I'd add automated source-credibility scoring. For now, I'm proud that our sponsor can build a briefing in minutes, not days—and that I helped turn an ambitious idea into a tangible, deployable solution.

Michael Zaino

My work spanned two distinct phases: an initial dashboard-focused approach (Part 1) and the subsequent development and documentation of the more integrated RENA system (Part 2). Lastly, I wrote the final report

Part 1: Initial Dashboarding & Analysis Concept

The first phase, documented in the Appendix 9, focused on broad data collection and visualization. Split into two main sections for me:

- **Diverse Web Scraping:** Implementing Python scripts using libraries like BeautifulSoup, direct API calls, and regular expressions to gather data from varied sources including Wikipedia, World Bank, Worldometer, and the CIA Factbook. This required adapting techniques to different website structures and data formats.
- **Data Processing:** Cleaning and structuring the scraped data into usable formats (JSON, CSV), handling missing values, and formatting numerical data were crucial steps.

This phase provided valuable experience in data scraping from multiple sources and the fundamentals of dashboard creation, also the limitations of maintaining certain aspects.

Part 2: RENA System Development

Building upon the learnings from Part 1, the RENA system(see more in Appendix 10) aimed for a more automated, integrated, and interactive solution focused on specific news sources (TWZ, Defense News). My coding work involved:

- **Focused Scraping (Article_Scraping.py):** Refining the scraping process to target specific article structures on TWZ and Defense News, improving robustness and data extraction accuracy for elements like full text, authors, and dates. Implementing duplicate checking against existing data.
- **ML Ranking Pipeline (ML_Ranking.py):** Implementing a practical ML pipeline using the transformers library.
- **API Development (api.py):** Building a backend API using FastAPI to serve data and handle user interactions.
- **Frontend Development (index.html):** Creating a user-facing web interface using HTML, CSS, and JavaScript.
- **System Integration & Setup:** Creating helper scripts (install_requirements.py, run_all.py) to streamline setup and execution.

This phase required integrating multiple components (scraping, ML, API, LLM, frontend) into a cohesive system, presenting challenges in managing dependencies, data flow, prompt engineering for the LLM, and ensuring smooth user interaction.

Overall Learning

This project provided extensive hands-on experience across the full stack of a data-driven application. Key learnings include advanced web scraping techniques, practical application of NLP models for classification and sentiment analysis, API development with FastAPI, integrating local LLMs using Ollama, building interactive frontends, and clear detailed documentation. The iterative documentation process also highlighted the value of breaking down complex systems into understandable parts. Lastly, remaking part 1 to be more robust after the development of RENA proved invaluable to ensure data collection proceeds smoothly.

Appendix 6: Completed Dashboards and RENA

Figure 1- Main dashboard 1

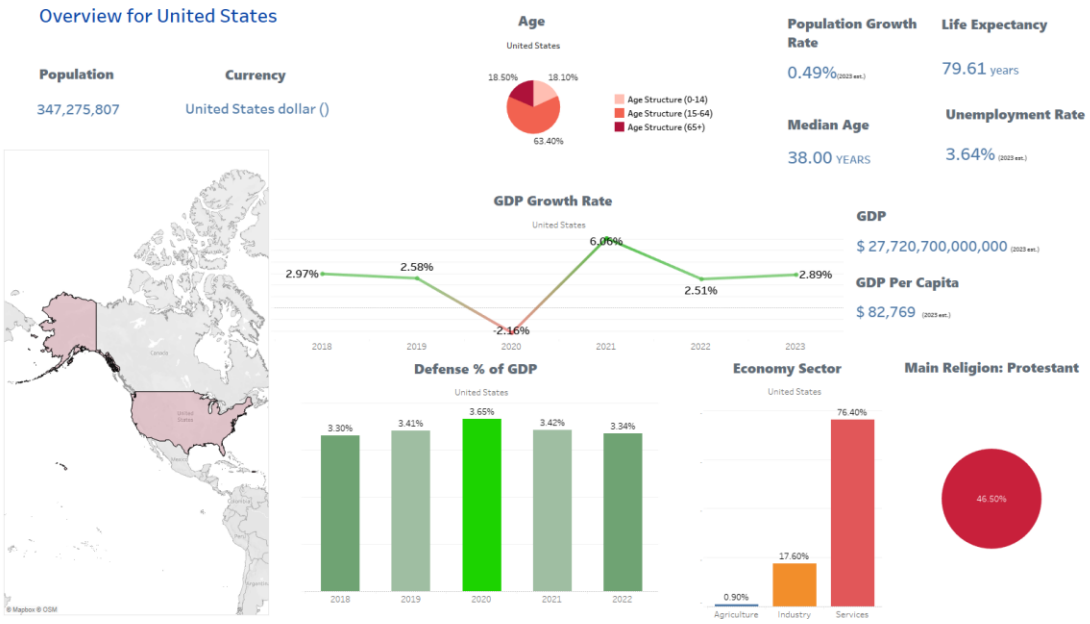


Figure 2- Main dashboard 2

Figure 3- Country map dashboard

Tank Strength	Towed Artillery	Rocket Projectors	Armored Fighting Vehicles	Self-Propelled Artillery	Army Personnel
521	480	54	31,964	2,220,000,000	196,700

Air Force Personnel	Helicopter Fleets	Trainer Fleets	Transports Fixed Wing	Attack Helicopters	Fighters / Interceptors
50,000	596	401	55	119	217

Tanker Fleets	Aircraft Fleet Strength	Attack/Strike
10	1,443	38

Helicopter Carriers	Corvettes	Mine Warfare Craft	Fleet Strength by Tonnage	Naval Fleet Strength	Destroyers
4	0	22	769,882	159	42

Frigates	Coastal Patrol Craft	Submarines	Navy Personnel
6	6	24	50,800

Fit-for-Service	Reaching Military Age	Active Service	Available Manpower	Active Reserves	Paramilitary
42,874,277	1,108,818	247,150	52,976,836	56,000	25,000

Total Population
123,201,945

Figure 4- GFP dashboard

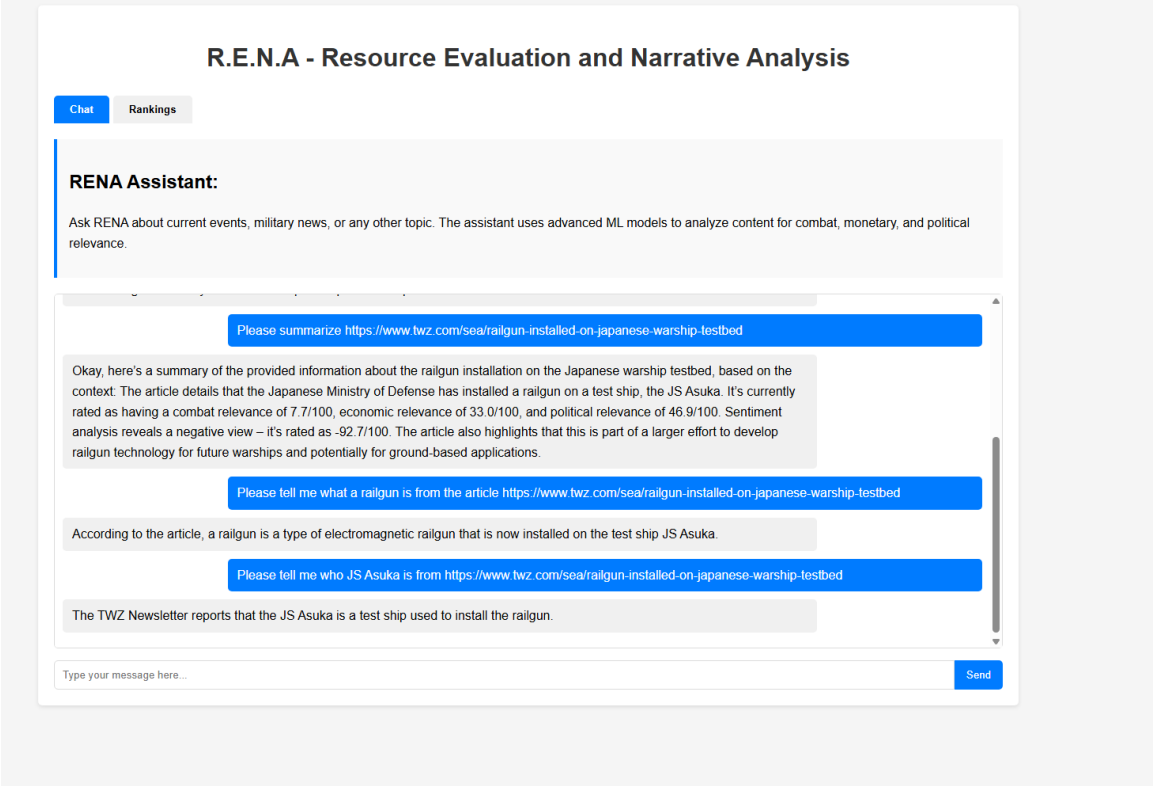

R.E.N.A - Resource Evaluation and Narrative Analysis
Chat
Rankings
RENA Assistant:
Ask RENA about current events, military news, or any other topic. The assistant uses advanced ML models to analyze content for combat, monetary, and political relevance.
Please summarize https://www.twz.com/sea/railgun-installed-on-japanese-warship-testbed
Okay, here's a summary of the provided information about the railgun installation on the Japanese warship testbed, based on the context: The article details that the Japanese Ministry of Defense has installed a railgun on a test ship, the JS Asuka. It's currently rated as having a combat relevance of 7.7/100, economic relevance of 33.0/100, and political relevance of 46.9/100. Sentiment analysis reveals a negative view – it's rated as -92.7/100. The article also highlights that this is part of a larger effort to develop railgun technology for future warships and potentially for ground-based applications.
Please tell me what a railgun is from the article https://www.twz.com/sea/railgun-installed-on-japanese-warship-testbed
According to the article, a railgun is a type of electromagnetic railgun that is now installed on the test ship JS Asuka.
Please tell me who JS Asuka is from https://www.twz.com/sea/railgun-installed-on-japanese-warship-testbed
The TWZ Newsletter reports that the JS Asuka is a test ship used to install the railgun.
Type your message here...
Send

Figure 5- R.E.N.A Web app visual

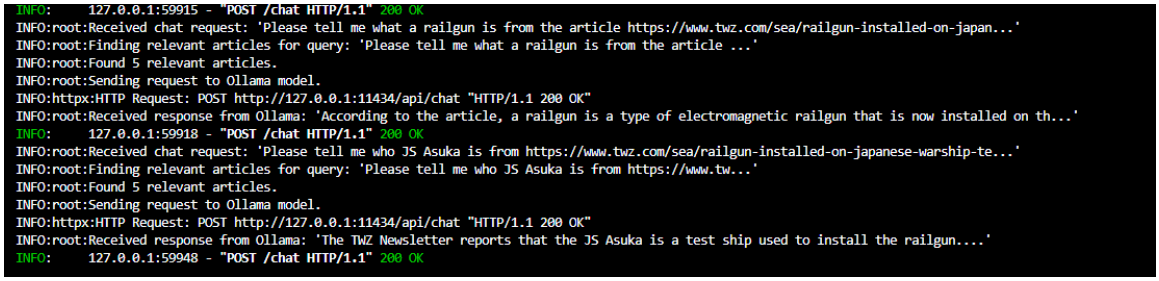

INFO: 127.0.0.1:59915 - "POST /chat HTTP/1.1" 200 OK
INFO:root:Received chat request: 'Please tell me what a railgun is from the article https://www.twz.com/sea/railgun-installed-on-japan...'
INFO:root:Finding relevant articles for query: 'Please tell me what a railgun is from the article ...'
INFO:root:Found 5 relevant articles.
INFO:root:Sending request to Ollama model.
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/chat "HTTP/1.1 200 OK"
INFO:root:Received response from Ollama: 'According to the article, a railgun is a type of electromagnetic railgun that is now installed on th...'
INFO: 127.0.0.1:59918 - "POST /chat HTTP/1.1" 200 OK
INFO:root:Received chat request: 'Please tell me who JS Asuka is from https://www.twz.com/sea/railgun-installed-on-japanese-warship-te...'
INFO:root:Finding relevant articles for query: 'Please tell me who JS Asuka is from https://www.tw...'
INFO:root:Found 5 relevant articles.
INFO:root:Sending request to Ollama model.
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/chat "HTTP/1.1 200 OK"
INFO:root:Received response from Ollama: 'The TWZ Newsletter reports that the JS Asuka is a test ship used to install the railgun....'
INFO: 127.0.0.1:59948 - "POST /chat HTTP/1.1" 200 OK

Figure 6- R.E.N.A backend working

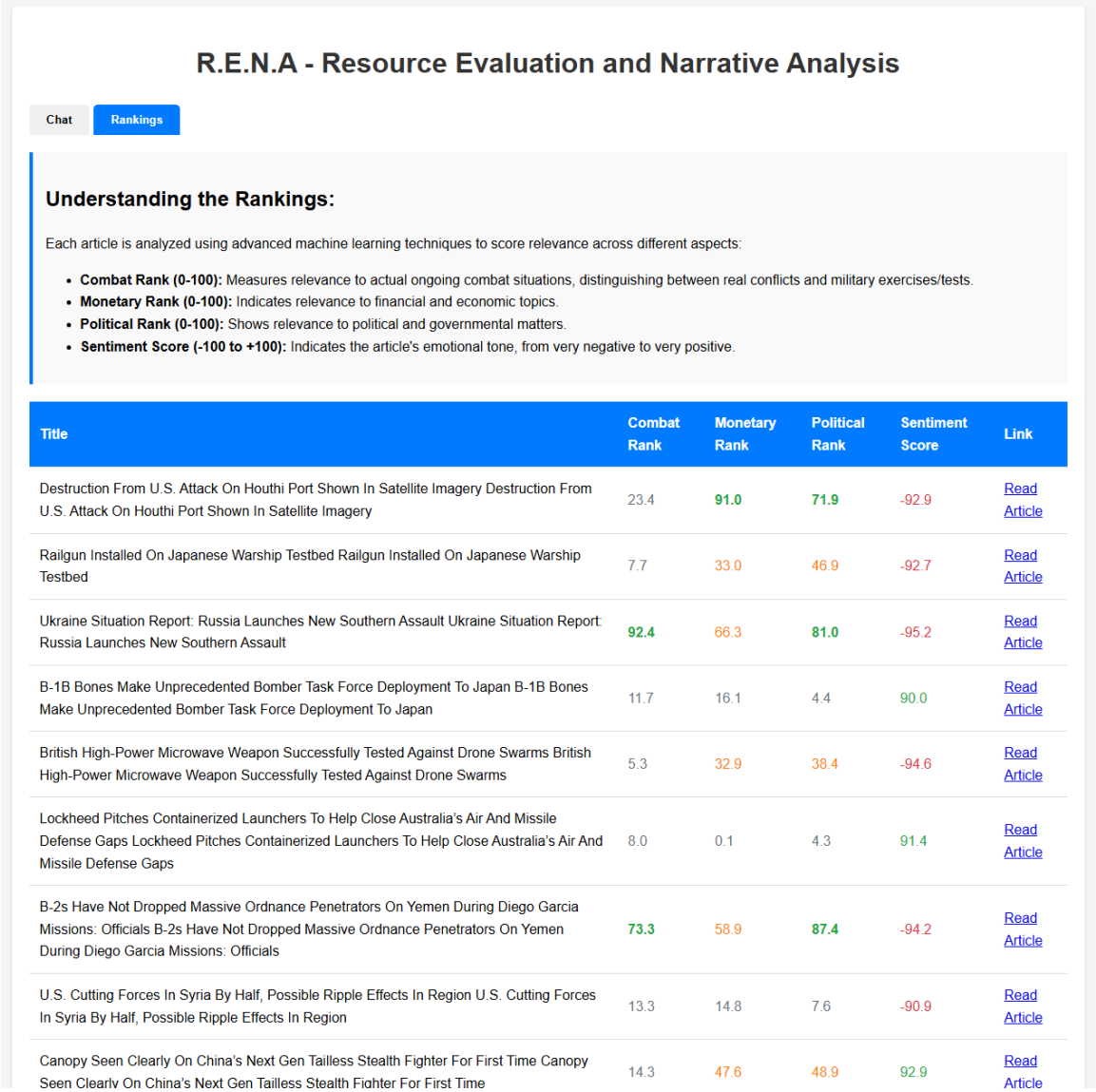

R.E.N.A - Resource Evaluation and Narrative Analysis
Chat
Rankings
Understanding the Rankings:
Each article is analyzed using advanced machine learning techniques to score relevance across different aspects:
Combat Rank (0-100): Measures relevance to actual ongoing combat situations, distinguishing between real conflicts and military exercises/tests.
Monetary Rank (0-100): Indicates relevance to financial and economic topics.
Political Rank (0-100): Shows relevance to political and governmental matters.
Sentiment Score (-100 to +100): Indicates the article's emotional tone, from very negative to very positive.
Title
Combat Rank
Monetary Rank
Political Rank
Sentiment Score
Link
Destruction From U.S. Attack On Houthi Port Shown In Satellite Imagery Destruction From U.S. Attack On Houthi Port Shown In Satellite Imagery
23.4
91.0
71.9
-92.9
Read Article
Railgun Installed On Japanese Warship Testbed Railgun Installed On Japanese Warship Testbed
7.7
33.0
46.9
-92.7
Read Article
Ukraine Situation Report: Russia Launches New Southern Assault Ukraine Situation Report: Russia Launches New Southern Assault
92.4
66.3
81.0
-95.2
Read Article
B-1B Bones Make Unprecedented Bomber Task Force Deployment To Japan B-1B Bones Make Unprecedented Bomber Task Force Deployment To Japan
11.7
16.1
4.4
90.0
Read Article
British High-Power Microwave Weapon Successfully Tested Against Drone Swarms British High-Power Microwave Weapon Successfully Tested Against Drone Swarms
5.3
32.9
38.4
-94.6
Read Article
Lockheed Pitches Containerized Launchers To Help Close Australia's Air And Missile Defense Gaps Lockheed Pitches Containerized Launchers To Help Close Australia's Air And Missile Defense Gaps
8.0
0.1
4.3
91.4
Read Article
B-2s Have Not Dropped Massive Ordnance Penetrators On Yemen During Diego Garcia Missions: Officials B-2s Have Not Dropped Massive Ordnance Penetrators On Yemen During Diego Garcia Missions: Officials
73.3
58.9
87.4
-94.2
Read Article
U.S. Cutting Forces In Syria By Half, Possible Ripple Effects In Region U.S. Cutting Forces In Syria By Half, Possible Ripple Effects In Region
13.3
14.8
7.6
-90.9
Read Article
Canopy Seen Clearly On China's Next Gen Tailless Stealth Fighter For First Time Canopy Seen Clearly On China's Next Gen Tailless Stealth Fighter For First Time
14.3
47.6
48.9
92.9
Read Article

Figure 7- R.E.N.A Rankings page

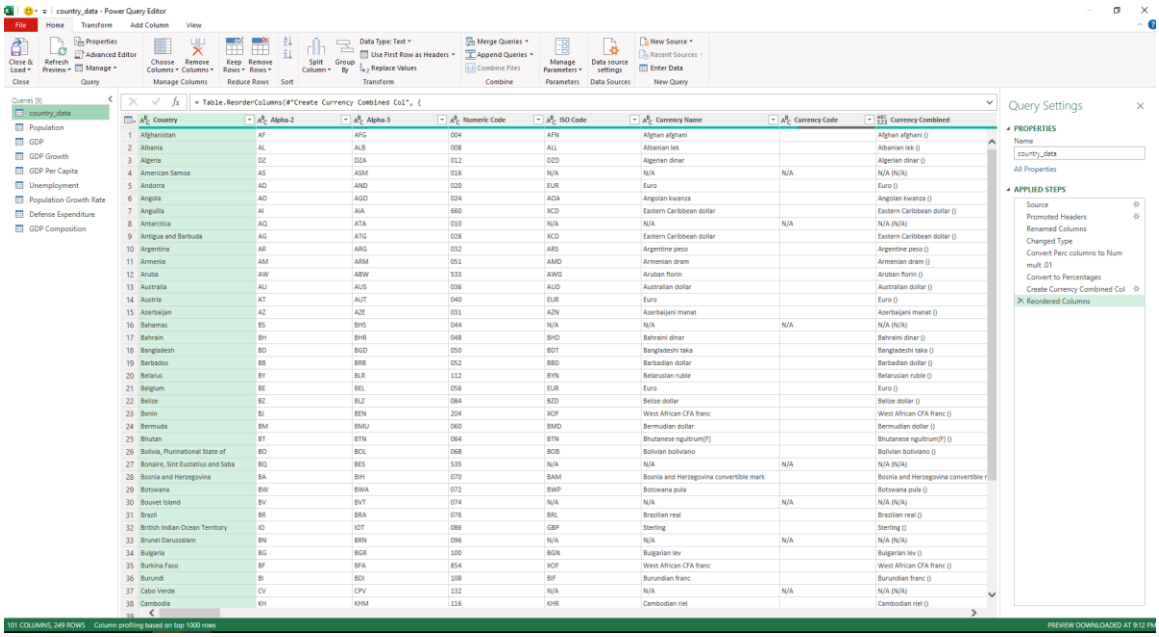


Figure 8- XLSX Cleaning file

Appendix 7: Future Development

- **Part 1 Enhancements:**
    - Integrate more data sources (trade data APIs, specific economic indicators).
    - Add historical data depth beyond the current 7-year window.
    - Develop more sophisticated visualizations or predictive analytics within Tableau.
    - Implement automated scheduling for the data pipeline (using Windows Task Scheduler or cloud functions).
- **Part 2 (RENA) Enhancements:**
    - **Vector Database:** Replace TF-IDF with semantic embeddings (Sentence-BERT) stored in a vector database (ChromaDB, Pinecone) for more accurate relevance search and scalability.
    - **Knowledge Extraction:** Implement NER and Relationship Extraction to build a knowledge graph from articles.
    - **Advanced RAG:** Refine the LLM interaction using more sophisticated Retrieval-Augmented Generation techniques.
    - **User Features:** Add user accounts, saved searches, customizable alerts, and document upload capabilities.
    - **Deployment:** Containerize (Docker) and explore cloud deployment options.

Appendix 8: Risk Management

- **Risk: Website Structure Changes (High Likelihood, High Impact):** Scrapers are inherently brittle. A change in a source website's HTML (class names, table structure) could break data extraction.
    - Mitigation: Used specific but relatively common HTML tags/IDs where possible; Implemented comprehensive try-except blocks around scraping logic to prevent script crashes and log errors; Developed modular scraper functions for easier updates; Advised sponsor on potential need for periodic maintenance.
- **Risk: Data Source Unavailability/API Changes (Medium Likelihood, High Impact):** A source website going offline, changing its URL structure, or altering its API (World Bank) could halt data collection.
    - Mitigation: Focused on established sources (World Bank, Factbook mirror, Worldometer); Implemented timeouts and status code checks for network requests; Designed the system to proceed with partial data if one source fails (logging the failure).
- **Risk: ML Model Accuracy/Bias (Medium Likelihood, Medium Impact):** Pre-trained models might exhibit biases or produce rankings/sentiments that don't align with expert assessment. LLMs can "hallucinate".
    - Mitigation: Used widely recognized models; Clearly communicated that ML scores are indicators, not definitive judgments; Designed RENA's chat to provide context from retrieved articles, grounding the LLM response rather than relying solely on its internal knowledge.

Appendix 9: Part1 documentation

PART 1 Documentation

Contents

Introduction 43

Methodology 43

How We Scrape It 43

Tableau Integration 44

Conclusion 45

Introduction

This document provides an overview of the web scraping process used for collecting and analyzing country intelligence data. It outlines the methodology, technical implementation, integration with Tableau.

Methodology

The project aims to automate the collection of economic, demographic, and defense-related data for countries. The process is defined from the following phases:

- **Data Collection:** Scraping data from multiple sources, including Wikipedia, World Bank, Worldometer, and CIA Factbook.
- **Data Cleaning:** Formatting numerical values, handling missing data, and structuring JSON outputs.
- **Data Storage & Output:** Storing data in JSON and CSV formats for further visualization and analysis.
- **Analysis & Insights:** Using Tableau for visualization and ALLAN for AI-driven textual insights.

How We Scrape It

Our web scraping process is built using Python with the following key libraries:

- **BeautifulSoup:** Extracts structured data from HTML pages.
- **API Calls:** Retrieves structured datasets from World Bank API.
- **Regular Syntax:** Cleans numerical and categorical data.

Key Functions in the Script:

- get_currency_info(): Extracts currency data from Wikipedia.
- get_world_bank_data(): Fetches economic indicators for countries.
- get_defense_expenditure(): Retrieves military spending data.
- get_worldometer_data(): Collects population and demographic statistics.
- get_factbook_data(): Extracts country intelligence from CIA Factbook.
- get_all_country_data(): Combines all the above functions to generate a complete dataset.

The final cleaned data is exported into structured CSV and JSON files for easy integration into Tableau.

Tableau Integration

Tableau is used to visualize the scraped and processed data. The integration process follows the order of:

- **Data Imported:** Processed CSV and JSON files are uploaded into Tableau.
- **Data Transformation:** Additional filtering and data formatting using Power Query.
- **Visualizations:**
    - Population and GDP comparisons across regions.
    - Inflation trends and defense expenditures over time.
    - Correlation analysis between economic and demographic indicators.
    - Map of current analyzed country.

Key Points in the Dashboards

- Size of Overview Tab → Built to be compatible mainly with (1600 x 1000)
    - Powerpoint (1600 x 900) will also work

- On the Overview Tab → Anywhere it says ('date' est) is manually written
    - Once the new years are released from scraped websites, you will have to manually change the date
        - **Population Growth Rate**
        - **Unemployment Rate**
        - **GDP**
        - **GDP Per Capita**

- On the Overview Tab → The map sometimes gets **locked** on a specific country
    - This is usually the last country you selected
    - To fix this, simply hover over the map and at the top left, click the thumbtack

- On the Overview Tab → The “Age” section of the dashboard sometimes reverts to not having the percentages shown
    - To fix this follow these steps
        - Select **All Countries** from the Physical Map Dashboard
        - Go to the Overview Tab → Right click in the Age box and **Select All**
        - Right click the small array of circles
        - Hover over **Mark Labels** and select **Always Show**

- On the Physical Map Dashboard → This utilizes **PictureThis**
    - This is an extension that can be downloaded via the “Marks” box in the dashboards
    - It should already have processed and all you need to do is download it once
        - This extension allows for importing images via links
        - We utilize GitHub and Git to complete this process once the files have been scraped
        - We will provide a Step-by-Step description below as well as a short video on how to upload via Git and how PictureThis pulls

Github and Git for Physical Map

Prerequisites: GitBash Download

Tentative List until we run through the process

1. cd to the file destination (Desktop)
2. git clone <url> #optional -b Branch
3. Go into the github repo
4. cp /path/to/your/files/* /path/to/Maps/
    a. You could also manually do this
5. git add .
6. git commit -m "Description of what you did"
7. git push origin main (main or whatever the name of the repo is that you cloned in)

Conclusion

This documentation outlines the technical aspects of our web scraping process, data visualization strategy, and AI-driven analysis. By combining automated scraping, utilizing

Tableau's visualization, and machine learning methods, the project delivers a comprehensive country intelligence data analyzation.

Appendix 10: RENA(Part2) Documentation

RENA DOCUMENTATION

Contents

Introduction 49
Methodology 49
Technical Implementation 49
API and User Interface (api.py & index.html) 55
System Setup & Execution 66
Future Work: 68
Conclusion 70

Introduction

This document provides an overview of the RENA (Resource Evaluation and Narrative Analysis) system. RENA is designed to automate the collection, analysis, and interpretation of news articles from specific defense and geopolitical sources. It scrapes relevant websites, applies machine learning models to rank articles based on predefined criteria (combat, monetary, political relevance) and sentiment, and provides an interactive chat interface for users to query the analyzed information using a large language model (LLM). The primary sources currently scraped are The War Zone (TWZ) and Defense News.

Methodology

The RENA project automates the workflow from raw web data to actionable insights through the following distinct phases:

- **Data Collection:** Automated scraping of articles from specified news websites (TWZ, Defense News) using Python scripts. Extraction of key metadata like title, author, date, section, and the full article text.
- **Data Storage:** Storing raw scraped article data persistently in a structured CSV format (articles.csv) to allow for incremental updates and prevent redundant scraping.
- **Data Processing & Ranking:** Applying machine learning models to newly scraped articles. This involves:
    - Zero-shot classification to determine relevance scores for combat, monetary, and political topics.
    - Sentiment analysis to gauge the overall tone of the article.
    - Storing these calculated ranks and scores in a separate CSV file (ranked_articles.csv), linked to the original articles.
- **Analysis & Interaction:**
    - Utilizing a FastAPI backend to serve the processed data.
    - Employing TF-IDF and Cosine Similarity to find articles relevant to user queries.
    - Integrating with the Ollama framework for local LLM (gemma3:1b) generation of responses based on the relevant articles.
    - Providing a web-based user interface (HTML/CSS/JS) for interacting with the chat assistant and viewing the ranked articles.

Technical Implementation

Web Scraping (Article_Scraping.py)

This script is responsible for automatically gathering article data from target news websites (The War Zone - TWZ, and Defense News). It simulates browser access to retrieve and parse web page content.

- **Libraries:** requests (sending HTTP requests to fetch web page HTML), BeautifulSoup4 (parsing HTML/XML, navigating structure via tags/attributes/selectors), pandas (reading/writing articles.csv for duplicate checks and data storage), csv (used by pandas for formatted CSV handling), os (checking articles.csv existence/size), re (regular expressions for text pattern matching), time (adding delays between requests to avoid server overload), logging (tracking progress and errors), urllib.parse (constructing absolute URLs from relative links using urljoin)

- **Core Process & Workflow:**

1. **Homepage Fetching:** Sends GET requests to the homepages of TWZ and Defense News using requests.get(), including the defined HEADERS and a timeout parameter to prevent the script from hanging indefinitely on unresponsive requests.

2. **Link Extraction:** Parses the homepage HTML using BeautifulSoup. It then searches for HTML anchor tags (<a>) that link to individual articles.

    - For TWZ (scrape_twz_articles): Looks for specific div elements with class card-post and extracts links from child <a> tags.
    - For Defense News (scrape_defensenews_articles): Looks for <a> tags whose href attribute matches a specific pattern (^/[^/]+/\d{4}/\d{2}/\d{2}/), typical for their article URL structure. urljoin is used to create full URLs.

3. **Article Iteration:** Loops through the extracted article links. For each unique link:

    - It calls functions to fetch the individual article page and extract detailed information.

4. **Duplicate Prevention:** Before saving, the script reads the 'Link' column from the existing articles.csv (if it exists) into an existing_links set. It only processes and prepares to save articles whose links are not in this set and were not already seen in the current scraping batch (seen_batch).

5. **Data Aggregation:** Stores the extracted data for each new article as a dictionary.

6. **Saving Data:** Appends the list of new article dictionaries to articles.csv using pandas.DataFrame.to_csv(). It ensures the header row is only written if the file is new or empty.

- **Key Helper Functions:**

    - clean_text(text):

- **Purpose:** A general utility function to standardize text extracted from HTML.
- **Action:** Takes a string, uses re.sub(r"\s+", " ", text) to replace sequences of one or more whitespace characters (spaces, tabs, newlines) with a single space, and then uses .strip() to remove leading/trailing whitespace. This ensures consistency and removes unwanted formatting.

- get_full_article_text(article_url):
  - **Purpose:** To retrieve the main textual content of an article, excluding headers, footers, ads, etc.
  - **Action:**
    1. Introduces a short delay (time.sleep(0.5)) to be polite to the server and prevent overloading requests.
    2. Fetches the content of the article_url using requests.get().
    3. Parses the HTML with BeautifulSoup.
    4. Attempts to find the main article content using specific CSS selectors known to work for the target sites (div.entry-content for TWZ, various selectors like div.article-content-body__content for Defense News). This targeted approach is more reliable than grabbing all text.
    5. If specific selectors fail, it falls back to finding all paragraph (<p>) tags within the <main> or <article> HTML5 semantic tags, or even the <body> as a last resort.
    6. Extracts the text from all found paragraph tags, cleans it using clean_text, and joins the paragraphs into a single string separated by spaces.
    7. Includes error handling (try...except) to catch request failures or parsing issues, logging the error and returning an empty string.
- extract_twz_date_info(date_string):
  - **Purpose:** Specifically designed to parse the date/time information as presented on TWZ articles.
  - **Action:** Takes the raw date string found on a TWZ card, cleans it, and uses regular expressions (re.search) to find patterns like "Posted" and "Updated.". It extracts the actual date/time part following these

keywords. Returns the posted and updated dates (or None if not found).

- extract_defensenews_details(article_url):
  - **Purpose:** To extract author and date information specifically from Defense News article pages, which can have varying structures.
  - **Action:**
    1. Introduces a delay (time.sleep(0.5)).
    2. Fetches and parses the individual article page.
    3. **Author Extraction:** Tries multiple CSS selectors (.author-text a, a[rel='author'], .byline a) known to contain the author's name, taking the first one found.
    4. **Date Extraction:** Attempts several methods to find the publication date:
       - Looks for <time> tags with specific attributes (itemprop='datePublished', class='published', dateti me attribute). Prioritizes the datetime attribute if available and attempts to parse it into a standard format using datetime.fromisoformat (handling timezone offsets).
       - If <time> tags fail or lack usable attributes, it searches the text content for patterns like "Published Month Day, Year" using re.search.
    5. Cleans the extracted author and date using clean_text.
    6. Returns the extracted author, posted date, and an empty string for updated date (as reliable updated dates weren't consistently found/targeted for DN in this script). Includes error handling.

- **Output:** Appends new, unique article data (Section, Title, Full_Article, Author(s), Date Posted, Date Updated, Updates/Summary, Link, Source) to the articles.csv file.

Machine Learning Ranking (ML_Ranking.py)

This script processes new articles found by the scraper and assigns relevance and sentiment scores using machine learning models.

- **Libraries:** pandas (data loading/saving), transformers (accessing pre-trained models from Hugging Face), nltk (sentence tokenization for sentiment

analysis), torch (ML framework and CUDA utilization), os(for interacting with the operating system for files and folders), logging(for monitoring).

- **Process:**

    1. **Setup:** Configures logging. Calls setup_nltk() to ensure the necessary 'punkt' tokenizer data is downloaded for sentence splitting. Checks for CUDA availability and sets the target device for model computations (GPU or CPU).
    2. **Data Loading:** Reads articles.csv into a DataFrame. Reads ranked_articles.csv if it exists; otherwise, initializes an empty DataFrame. Creates a set (seen_links) of article links already present in ranked_articles.csv.
    3. **New Articles:** Identifies articles present in articles.csv but not yet in ranked_articles.csv.
    4. **Model Initialization:**
        - Initializes the zero-shot-classification pipeline using the facebook/bart-large-mnli model. The device parameter ensures the model is loaded onto the appropriate hardware (GPU/CPU). This model is chosen for its strong performance on Natural Language Inference (NLI), and more importantly it's zero-shot classification.
        - Initializes the sentiment-analysis pipeline, also specifying the device
    5. **Prepare Texts:** Creates a list of texts for classification by combining the 'Title' and the first 1000 characters of the 'Full_Article' for each new article (get_classification_text). Truncating the text helps manage computational load while usually retaining enough context for classification.
    6. **Batch Classification:** Processes the prepared texts in batches (size 8, adjustable) using the zero-shot classifier. Batching improves efficiency, especially on GPUs. The classifier evaluates each text against the predefined labels. Error handling is included for this step.
    7. **Sentiment Analysis & Result Aggregation:** Iterates through the classification results and the corresponding new article data:
        - Extracts the scores for each predefined label (combat, monetary, political) from the classification output.
        - Calls the analyze_sentiment function for the full article text to get the sentiment score.
        - Stores the title, link, calculated ranks (probabilities * 100), and sentiment score in a list of dictionaries (new_results).
    8. **Output**: Appends the title, link, calculated ranks (combat_rank, monetary_rank, political_rank), and sentiment_score for new articles to ranked_articles.csv

- **Key Helper Functions & Components:**
    - **setup_nltk():**
        - **Purpose:** Ensures the NLTK 'punkt' resource (pre-trained sentence boundary detection model) is available**.**
        - **Process:** Uses nltk.data.find to check if 'punkt' is accessible. If not (LookupError), it calls nltk.download('punkt') to fetch and install it. This is necessary for the sent_tokenize function used in sentiment analysis.
    - **get_classification_text(title, full_text):**
        - **Purpose:** Creates a concise input string suitable for the zero-shot classifier.
        - **Process:** Concatenates the article title with the first 1000 characters of the full article text. This provides key topic information (from the title) and initial context (from the article body) without overwhelming the model.
    - **Sentiment Analysis:**
        - **Model:** Uses a standard sentiment-analysis pipeline from transformers.
        - **Preprocessing:** The article text is first tokenized into individual sentences using NLTK's sent_tokenize (based on the Punkt algorithm for boundary detection).
        - **Math/Concept:** The pipeline analyzes each valid sentence (longer than 10 characters) and outputs a label ('POSITIVE' or 'NEGATIVE') and a confidence score (probability, 0 to 1).
        - **Score Calculation:** The script calculates an overall sentiment score by:
            1. Averaging the confidence scores of all analyzed sentences: avg_score = sum(sentence_scores) / num_sentences.
            2. Determining the dominant sentiment by counting 'POSITIVE' vs 'NEGATIVE' labels.
            3. If 'POSITIVE' count > 'NEGATIVE' count, the final score is avg_score * 100.
            4. If 'NEGATIVE' count >= 'POSITIVE' count, the final score is -avg_score * 100.

5. This results in a score ranging approximately from -100 (very negative) to +100 (very positive).

- **Zero-Shot Classification:**
  - **Model:** Uses the facebook/bart-large-mnli model via the transformers pipeline. This model is based on the BART transformer architecture.
  - **BART:** BART is a transformer model designed for sequence-to-sequence tasks. It's pre-trained on large text corpora and then fine-tuned on the Multi-Genre Natural Language Inference (MNLI) dataset. In other words, BART is originally trained on a large collection of texts which is then fine-tuned for better results.
  - **Mechanism (NLI):** The pipeline reframes classification as NLI. For a label like "economic and financial topics", it creates a hypothesis ("This text is about economic and financial topics"). It then feeds the model the article snippet (premise) and the hypothesis. The model outputs probabilities for three NLI labels: 'entailment', 'contradiction', and 'neutral'. The probability of 'entailment' is used as the score for the candidate label. This allows classification using labels the model wasn't explicitly trained on, hence "zero-shot".
  - **Math/Concept:** Zero-shot classification is performed by treating the task as Natural Language Inference (NLI). The model assesses the likelihood that the article text (premise) entails a hypothesis(this means that if premise is true, then the hypothesis must also be true) constructed from the candidate labels. The raw output for each label is a probability score (between 0 and 1) representing this entailment likelihood.
  - **Ranking:** Classifies each new article against predefined labels:
    1. "actual military combat and conflicts" (combat_rank)
    2. "economic and financial topics" (monetary_rank)
    3. "political and governmental matters" (political_rank)
  - **Score Calculation:** The entailment probability for each label is multiplied by 100 to produce the final rank (0-100 scale). Rank = P(premise entails hypothesis_label) * 100

API and User Interface (api.py & index.html)

- **Backend (api.py):**

- **Framework:** Built using FastAPI.
- **Libraries:** pandas (data loading/merging), scikit-learn (TF-IDF Vectorizer, Cosine Similarity), ollama (interface with local LLM), uvicorn (ASGI server), jinja2 (HTML templating), numpy(Array manipulation), logging(for monitoring).
- **Core Process & Workflow:**

    1. **Initialization:** Sets up logging and creates the FastAPI application instance (app). Configures the Jinja2 template directory.
    2. **Data Loading (on startup):** Calls load_data() to read and prepare the article and ranking data.
    3. **NLP Setup (on startup):** Calls initialize_nlp() to create the TF-IDF vectorizer and matrix based on the loaded data.
    4. **Server Start:** The script, when run directly (if __name__ == "__main__":), uses uvicorn.run() to start the ASGI server, making the API accessible over the network (typically at http://localhost:8000).
    5. **Request Handling:** Listens for incoming HTTP requests to the defined endpoints (/ and /chat) and executes the corresponding functions (read_root, chat_endpoint).

- **Functionality:**

    - load_data():
      Loads and merges data from articles.csv and ranked_articles.csv on startup.

        - **Process:**

        1. Attempts to read articles.csv and ranked_articles.csv using pd.read_csv.
        2. Renames the 'Link' column in articles.csv to 'link' for consistency.
        3. Standardizes the 'link' column in both DataFrames (stripping whitespace, lowercasing) to ensure accurate merging.
        4. If both DataFrames are loaded successfully, it performs a left merge (pd.merge(..., how='left')) using the 'link' column as the key (via temporary 'link_key' columns). This combines ranking information with the corresponding article details. It handles potential column name conflicts using suffixes and cleans up redundant columns post-merge.

5. If only one file is found, it returns that DataFrame. If neither is found, it returns an empty DataFrame.

6. Ensures essential columns (Title, Full_Article, etc.) exist in the final DataFrame, adding them with None values if missing, preventing errors later.

- initialize_nlp():
  Initializes a TF-IDF vectorizer based on the combined title and text of the articles.

  - **Process:**

  1. Checks if the combined_df (from load_data) is not empty and contains text data ('Full_Article').

  2. Safely combines the 'Title' and 'Full_Article' columns into a list of text documents, handling potential NaN values using .fillna('').

  3. If text data exists, it initializes a TfidfVectorizer from scikit-learn. stop_words='english' removes common English words that don't usually contribute to topic relevance.

  4. Calls tfidf.fit_transform(texts): This step is crucial as it performs two actions:

     - fit: Learns the vocabulary of the entire article corpus and calculates the IDF for each term.

     - transform: Converts the list of text documents into the tfidf_matrix, a sparse matrix where rows represent documents and columns represent terms, containing the TF-IDF scores.

  5. Logs the shape of the created matrix and returns the fitted tfidf object (containing vocabulary and IDF values) and the tfidf_matrix.

  6. Handles potential errors during initialization and returns None, None if setup fails or no text data is available.

- find_relevant_articles(query, top_n=5):
  To identify the top_n articles from the corpus that are most semantically similar to the user's input query

  - **Process:**

  1. Checks if the TF-IDF model (tfidf, tfidf_matrix) was successfully initialized. Returns an empty DataFrame if not.

2. Uses the already fitted tfidf object's transform([query]) method to convert the user's query string into a TF-IDF vector using the vocabulary learned from the articles. This ensures the query is represented in the same vector space as the documents.
3. Calculates the cosine_similarity between the single query vector and all document vectors in the tfidf_matrix. This results in an array of similarity scores (0 to 1), one for each document.
4. Determines the actual number of top articles to retrieve (actual_top_n), ensuring it doesn't exceed the total number of articles.
5. Efficiently finds the indices of the actual_top_n highest similarity scores. It uses np.argpartition if top_n is relatively small compared to the total number of articles (faster for finding a small number of top elements without fully sorting), otherwise uses np.argsort (full sort).
6. Sorts these top indices based on their corresponding similarity scores in descending order to get the final ranked list of indices.
7. Uses these indices (top_indices_sorted) to select and return the relevant rows from the original combined_df.
8. Includes error handling for the relevance calculation process.

- **Math/Concept (TF-IDF):** Term Frequency-Inverse Document Frequency is a numerical statistic reflecting a word's importance to a document within a collection (corpus).
    - **Term Frequency (TF):** Measures how frequently a term “t” appears in a document “d”. Calculated as:
        - TF(t, d) = (Number of times term “t” appears in document “d”) / (Total terms in document “d”)
    - **Inverse Document Frequency (IDF):** Measures how common or rare a term “t” is across the entire corpus “D”. Calculated as:
        - IDF(t, D) = log((Total number of documents |”D”|) / (Number of documents containing term “t” + 1))
        - The logarithm dampens the effect of high counts, and +1 prevents division by zero.
    - **TF-IDF Score:** The final score combines both measures:
        - TF-IDF(t, d, D) = TF(t, d) * IDF(t, D)

- **Vectorization:** The fit_transform method calculates the TF-IDF scores for all terms in all documents, creating a document-term matrix (tfidf_matrix) where each row is a document represented as a vector of TF-IDF scores.

- **/ Endpoint:** Serves the main HTML page (index.html) using Jinja2 templates, passing the ranked article data for display.
  - @app.get("/", response_class=HTMLResponse) - async def read_root(request: Request): Handles GET requests to the root URL (/). This is the entry point for users accessing the web interface.

- **/chat Endpoint:**
  - @app.post("/chat") - async def chat_endpoint(request: Request, user_input: str = Form(...)): Handles POST requests sent from the chat interface when the user submits a message.
  - **Process:**
    1. Receives the user_input from the submitted form data.
    2. Calls find_relevant_articles() with the user_input to get the most similar articles.
    3. Constructs the context string by iterating through the relevant articles, extracting titles, ranks, snippets, and links. Handles cases where ranking data might be missing (pd.notna).
    4. Defines the system_prompt to guide the LLM's behavior.
    5. Packages the system_prompt, the generated context, and the user_input into the message structure required by the ollama.chat function.
    6. Calls ollama.chat(model='gemma3:1b', messages=...) to send the request to the Ollama service.
    7. Extracts the content of the LLM's response from the returned dictionary.
    8. Returns the AI's response as a JSON object {"response": bot_response} back to the frontend JavaScript.
    9. Includes error handling for communication issues with Ollama.

- **Relevance Calculation:**

i. Transforms the user's query into a TF-IDF vector using the already fitted vectorizer (tfidf.transform([query])).

ii. **Math/Concept (Cosine Similarity):** Calculates the similarity between the user's query vector and each document vector in the tfidf_matrix. Cosine similarity measures the cosine of the angle between two vectors, indicating their orientation similarity irrespective of magnitude.

1. Formula: Similarity(QueryVector, DocVector) = (QueryVector · DocVector) / (||QueryVector|| * ||DocVector||)
2. Where · denotes the dot product, and ||V|| denotes the Euclidean norm (magnitude) of vector V.
3. The result ranges from 0 (no similarity) to 1 (identical orientation).

iii. Uses cosine_similarity from scikit-learn to efficiently compute this between the query vector and all document vectors.

iv. Identifies the indices of the documents with the highest cosine similarity scores (top_n articles).

- Constructs a context string containing titles, key ranks, snippets, and links from these most relevant articles.
- Sends this context along with the user's original query and a system prompt (defining the AI's persona and task) to Ollama Gemma3:1b.
- **LLM Interaction via Ollama:**
- What is an LLM?
  A Large Language Model (LLM) is a type of artificial intelligence model trained on massive amounts of text data. It learns patterns, grammar, and context, enabling it to understand and generate human-like text. Its core function is often predicting the next word (or token) in a sequence, allowing it to complete sentences, answer questions, summarize text, and perform various other language tasks. The key aspect here is the use of matrix multiplication to process and generate responses. Matrix multiplication is done typically by the GPU which is why we utilized CUDA(Compute Unified Device Architecture) which is how NIVIDIA GPUs to do parallel computation for matrix multiplication, thus far exceeding CPU performance. RENA uses the gemma3:1b model specifically, where 1b means that this model was trained on 1 billion parameters.

- What is Ollama?
  Ollama is a tool or framework that simplifies the process of downloading, setting up, and running various open-source LLMs (like Gemma, Llama, Mistral) directly on a user's local machine (CPU or GPU). Specifically in this case Gemma3:1b. It manages the models and provides a standardized API endpoint (typically http://localhost:11434) for applications to interact with them.
- How RENA uses them: The api.py script uses the ollama Python library to communicate with the locally running Ollama service. It sends a structured prompt containing:
  1. A system_prompt defining the AI's persona (RENA) and task.
  2. The dynamically generated context (from relevant articles found via TF-IDF/Cosine Similarity).
  3. The original user_input.
     Ollama receives this request, passes it to the specified LLM (gemma3:1b), which then processes the entire input and generates a text response based on its training and the provided context.
- Returns the LLM's generated response in JSON format.

- **Frontend (index.html):**
  - **Technology:** Technology: Standard HTML for structure, embedded CSS within <style> tags for presentation, and vanilla JavaScript for interactivity.
  - **Structure:** Contains a main div.container holding the title, tab buttons, and tab content areas (div#chat, div#rankings). Includes explanatory text sections (div.explanation) for both tabs.
  - **Structure:** Contains a main div.container holding the title, tab buttons, and tab content areas (div#chat, div#rankings). Includes explanatory text sections (div.explanation) for both tabs.
  - **Styling (CSS):**
    - The embedded CSS defines the overall layout (using flexbox for tabs and chat input), fonts, colors, padding, borders, and visual feedback (hover effects on table rows and buttons).
    - **Conditional Coloring:** Specific CSS classes are defined to visually represent rank levels and sentiment scores:
      - .score-high (Green, bold), .score-medium (Orange), .score-low (Gray) for ranks.
      - .sentiment-positive (Green), .sentiment-negative (Red), .sentiment-neutral (Gray) for sentiment.

        - These classes are dynamically applied to the table cells (<td>) within the Jinja2 template based on the numerical value of the scores ({% if article.combat_rank > 70 %}score-high{% elif ... %}).
    - **Chat Appearance:** Classes .user-message (blue background, white text) and .assistant-message (light gray background) differentiate between user and AI messages in the chat window. These classes are assigned via JavaScript when messages are added.
    - **Active Tab:** The .tab-button.active class (blue background, white text) highlights the currently selected tab button, managed by the showTab JavaScript function.

- **Features:**
    - **Tab Navigation:**
        - The showTab(tabId) JavaScript function handles switching between the "Chat" and "Rankings" views.
        - It hides all div.tab-content elements and then displays the one matching the tabId.
        - It updates the visual state of the button.tab-button elements, adding the active class to the selected tab's button.
    - **Rankings Tab:** Displays the ranked articles loaded from ranked_articles.csv via the backend. Includes sortable columns for Title, Ranks, and Sentiment. Provides direct links to the source articles. Explains the meaning of the ranks:
        - Combat Rank (0-100) - Measures relevance to actual ongoing combat situations, distinguishing between real conflicts and military exercises/tests.
        - Monetary Rank (0-100) - Indicates relevance to financial and economic topics.
        - Political Rank (0-100) - Shows relevance to political and governmental matters.
        - Sentiment Score (-100 to +100) - Indicates the article's emotional tone, from very negative to very positive.
    - Contains an HTML <table> (id="articlesTable") to display ranked articles.
    - Data Population: The table body (<tbody>) is populated server-side by the FastAPI backend using Jinja2 templating ({% for article in ranked_articles %} loop), iterating through the data passed from api.py. Jinja filters (|round(1)) are used for formatting scores.

- Sorting: Table headers (<th>) have an onclick event calling the sortTable(columnIndex) JavaScript function. This function implements client-side sorting:
    - It determines the column type (numeric for ranks/sentiment, text otherwise).
    - It sorts the table rows based on the selected column's content.
    - It toggles between ascending (asc) and descending (desc) sort directions.
    - It adds visual indicators (↑ or ↓) to the sorted column header.
- **Chat Tab:**
    - Features a chat display area (div#chatMessages) and an input form (form#chatForm) with a text input (input#userInput) and send button.
    - Interaction (sendMessage(event) function):
        - Triggered on form submission; event.preventDefault() stops the default page reload.
        - Retrieves the user's text from the input field.
        - Dynamically creates a new div.message.user-message element, sets its text content, and appends it to the chatMessages area.
        - Clears the input field.
        - Uses the fetch API to send the user's message asynchronously to the /chat endpoint on the backend (POST request, URL-encoded data).
        - Handles the JSON response from the backend: creates a div.message.assistant-message, populates it with the AI's response (data.response), and appends it to the chat.
        - Includes basic error handling to display a message if the fetch call fails.
        - Automatically scrolls the chatMessages area to the bottom after adding new messages.

- **JavaScript Functions & Interactivity(indepth):**

- **showTab(tabId):**
  This manages the display of content tabs ("Chat" and "Rankings") and update the visual state of the tab buttons.
  - **Trigger:** Called via the onclick event handler attached to each tab button (<button class="tab-button" onclick="showTab('chat')">). The tabId argument ('chat' or 'rankings') is passed based on which button was clicked.
- **Process:**
  1. **Hide All Tabs:** Selects all elements with the class tab-content using document.querySelectorAll('.tab-content') and iterates through them using forEach, removing the active class from each. This ensures only the desired tab will be visible.
  2. **Show Selected Tab:** Selects the specific tab content element by its ID (document.getElementById('chat')) using the passed tabId and adds the active class to it. The CSS rule .tab-content.active { display: block; } makes this element visible (while .tab-content { display: none; } hides the others).
  3. **Deactivate All Buttons:** Selects all elements with the class tab-button and removes the active class from each, resetting their appearance.
  4. **Activate Clicked Button:** Finds the specific button that was clicked (by checking which button's onclick handler contains the tabId) and adds the active class to it. The CSS rule .tab-button.active styles this button differently (blue background, white text).
- **sortTable(columnIndex):**
  This will sort the rows of the "Rankings" table (id="articlesTable") based on the data in a specific column, allowing users to order articles by different criteria (Title, Ranks, Sentiment).
  - **Trigger:** Called via the onclick event handler attached to each table header cell (<th>) in the "Rankings" table (<th onclick="sortTable(0)">Title</th>). The columnIndex argument indicates which column header was clicked (0 for the first column, 1 for the second, etc.).
- **Process:**
  1. **Initialization:** Gets references to the table (articlesTable), initializes variables for the sorting loop (switching, rows, shouldSwitch, switchcount), and sets the initial sort direction (dir = "asc").
  2. **Sorting Loop (while (switching)):** Continuously iterates through the table rows until no more swaps are needed in a pass for the current direction.
  3. **Row Comparison:** In each pass, it iterates through the table rows (for (i = 1; i < (rows.length - 1); i++)), comparing the content of the

cell in the columnIndex of the current row (x) with the cell in the same column of the next row (y).

4. **Type Handling:** Checks if the columnIndex corresponds to a numeric column (ranks/sentiment, indices 1-4). If so, it parses the cell text content into floating-point numbers (parseFloat) for numerical comparison. Otherwise, it performs a case-insensitive string comparison (toLowerCase()).
5. **Switch Decision:** Based on the current sort direction (dir) and the comparison result (xValue > yValue for ascending numeric), it sets shouldSwitch = true if the rows are out of order.
6. **DOM Manipulation (Row Swap):** If shouldSwitch is true, it physically moves the next row (rows[i + 1]) before the current row (rows[i]) in the HTML structure using rows[i].parentNode.insertBefore(rows[i + 1], rows[i]). This visually reorders the rows in the browser. It sets switching = true to indicate a swap occurred and increments switchcount.
7. **Direction Toggle:** If a full pass completes with no swaps (switchcount === 0) and the direction was ascending (dir === "asc"), it flips the direction to descending (dir = "desc") and sets switching = true to run the loop again in the new direction.
8. **Header Update:** After the sorting loop finishes, it iterates through all table headers (<th>), removes any existing sort indicator arrows (↑ or ↓), and then appends the appropriate arrow to the header of the column that was just sorted, indicating the final sort direction.

- **sendMessage(event):**
  This handle the submission of the user's message in the chat interface, send it to the backend API, and display both the user's message and the AI's response in the chat window.
  - **Trigger:** Called via the onsubmit event handler attached to the chat input form (<form id="chatForm" onsubmit="sendMessage(event)">).
- **Process:**
  1. **Prevent Default Submission:** event.preventDefault() stops the browser's default form submission behavior, which would normally cause a full page reload. This allows for handling the submission asynchronously using JavaScript.
  2. **Get User Input:** Retrieves the text entered by the user from the input field (id="userInput") and removes leading/trailing whitespace using .trim().
  3. **Display User Message:** If the input text is not empty:
     - Selects the chat messages container (id="chatMessages").
     - Creates a new div element using document.createElement('div').
     - Assigns the appropriate CSS classes (message user-message) for styling.

- Sets the textContent of the new div to the user's message.
- Appends the new div to the chatMessages container using appendChild.

4. **Clear Input:** Resets the value of the input field (userInput.value = '';).
5. **Scroll Chat:** Scrolls the chatMessages container to the bottom (chatMessages.scrollTop = chatMessages.scrollHeight) so the latest message is visible.
6. **Send to Backend (Fetch API):**
   - Uses the browser's fetch API to make an asynchronous HTTP POST request to the /chat endpoint of the backend API.
   - Sets the method to 'POST'.
   - Sets the Content-Type header to application/x-www-form-urlencoded as expected by FastAPI's Form parameter handling.
   - Constructs the request body by URL-encoding the user's input (user_input=${encodeURIComponent(messageText)}).
7. **Handle Response:** Uses promise chaining (.then(...)) to handle the asynchronous response:
   - The first .then parses the JSON response body using response.json().
   - The second .then receives the parsed data object (containing the response field from the backend). It then:
     - Creates another new div element.
     - Assigns the appropriate CSS classes (message assistant-message).
     - Sets its textContent to the AI's response (data.response).
     - Appends this AI message div to the chatMessages container.
     - Scrolls the chat to the bottom again.
8. **Error Handling:** Includes a .catch(error => ...) block to handle potential network errors or issues during the fetch request. If an error occurs, it logs the error to the console and displays a generic error message in the chat window.

System Setup & Execution

To run the RENA system, follow these steps:

1. **Prerequisites:**
   - **Python:** Ensure Python 3.8 or later(I used 3.11.9) is installed.
   - **Git:** (Optional, but recommended for cloning) Install Git.
   - **Ollama:**
     1. Install Ollama for your operating system from https://ollama.com/download.

2. Pull the required LLM model by running the following command in your terminal:
    1. ollama pull gemma3:1b
3.

- Verify Ollama is running in the background(bottom right corner click the little arrow, you should see the Ollama logo). Also can use “ollama list”
- CUDA: (Optional, for GPU acceleration in ML_Ranking.py) Ensure appropriate NVIDIA drivers and CUDA toolkit compatible with PyTorch (cu118 specified in install_requirements.py) are installed if you have a compatible GPU.

2. **Install Dependencies(Can be done in virtual environment)**
    - **For Virtual Environment Do This First:** Using a virtual environment prevents conflicts between project dependencies and your global Python installation.
        1. python -m venv venv
        2. .\venv\Scripts\activate (For windows)
        3. source venv/bin/activate (For linux/Mac)
    - Run the installation script (this handles PyTorch/CUDA correctly as default is cpu only):
        1. python install_requirements.py(or run in vscode)
    - Alternatively, install directly from the file (ensure PyTorch+CUDA is handled if needed):
        1. pip install -r requirements.txt
        2. Potentially install torch separately for CUDA:
        3. pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
    - Download NLTK Data (if first time):
        1. The ML_Ranking.py script attempts to download the 'punkt' tokenizer data automatically if needed. If this fails, you might need to run Python and execute:
        2. import nltk
        3. nltk.download('punkt')

3. **Run the System:**

- You can run the components individually or use the provided run_all.py script (after uncommenting the desired steps).
- Option A: Run Sequentially (Recommended for first run/updates)
    1. python Article_Scraping.py (Scrape for new articles)
    2. python ML_Ranking.py (Rank the new articles)
    3. python api.py (Start the web server)
- Option B: Use run_all.py (Ensure steps are uncommented as needed)
    1. python run_all.py
    2. Note: As provided, run_all.py has scraping and ranking commented out. Uncomment them if you want it to perform those steps before starting the API.

4. **Accessing the Interface:**
    - Once the api.py script (or run_all.py including the API step) is running, it will output messages indicating the server has started
    - Open your web browser and navigate to the address shown, usually: http://localhost:8000
    - You should see the RENA web interface with the "Chat" and "Rankings" tabs.
5. **Stopping the System:**
    - To stop the web server (api.py or run_all.py), go back to the terminal where it is running and press CTRL+C.
    - Remember to stop the Ollama application/service if you no longer need it running.
    - To deactivate the virtual environment, simply type: “deactivate”

Future Work:

- **Expand Data Sources & Input:**
    - Integrate scrapers for additional relevant news sources, blogs, official government/military press releases, or academic journals.
    - Develop methods to handle websites requiring JavaScript rendering.
    - Implement a configuration system to easily add/remove target websites.
    - **User Document Upload:** Allow users to upload their own documents (PDF, DOCX, TXT) for analysis. These could be summarized, ranked against the

system's criteria, compared for similarity against the scraped corpus, or used as context for LLM queries.

- **Enhance ML Models & Analysis:**
    - Experiment with more advanced or fine-tuned models for classification, sentiment analysis, and potentially Named Entity Recognition (NER) to extract key entities (people, places, organizations, equipment).
    - Introduce additional ranking criteria: source credibility scoring, article timeliness/recency weighting, identifying specific entities or geopolitical events.
    - Explore unsupervised techniques like Topic Modeling (LDA, BERTopic) to automatically discover emerging themes.
    - Fine-tune a smaller LLM specifically on the domain data for potentially faster and more tailored chat responses.
    - **Advanced Summarization:** Implement more sophisticated summarization techniques (extractive or abstractive) beyond providing fixed-length snippets in the chat context.
- **Improve User Interface & Experience:**
    - Enhance the "Rankings" tab with filtering (by source, date range, score thresholds), text search, and potentially keyword highlighting.
    - Integrate basic data visualizations directly into the web interface (score trends, topic distributions).
    - Implement user accounts for personalized settings, saved queries, or managing uploaded documents.
    - Refine the chat interface: support conversational history, allow editing previous prompts, provide options to adjust context length sent to the LLM.
    - **Real-time Alerts:** Implement a notification system to alert users about newly scraped articles matching predefined keywords or relevance criteria.
- **Strengthen Backend & Infrastructure:**
    - **Vector Database Integration:** Replace the in-memory TF-IDF matrix and cosine similarity calculation with a dedicated vector database (ChromaDB, Pinecone). This would:
        - Allow for much larger datasets of articles.

        - Enable more efficient and scalable semantic similarity searches using embeddings (potentially from models like Sentence-BERT instead of TF-IDF).
        - Facilitate storing and searching user-uploaded documents alongside scraped articles.
    - Replace CSV file storage with a relational or NoSQL database (PostgreSQL, SQLite, MongoDB) for structured metadata, better querying, and scalability.
    - Implement caching mechanisms (Redis) for API responses, LLM queries, or computed results.
    - Containerize the application using Docker/Docker Compose for easier setup, deployment, and environment consistency.
    - Explore cloud deployment options (AWS, Google Cloud, Azure) for scalability and accessibility.
- **Refine LLM Interaction & Knowledge Representation:**
    - Implement more sophisticated prompt engineering and potentially Retrieval-Augmented Generation (RAG) techniques leveraging the vector database for more accurate context injection.
    - Add conversational memory to the chat.
    - Allow users to select different LLMs available through Ollama.
    - **Knowledge Graph Integration:** Explore extracting entities and relationships from articles to build a knowledge graph, enabling more complex queries about connections between events, people, and organizations.

Conclusion

RENA provides a framework for automating the collection and analysis of specialized news content. By combining web scraping, machine learning for relevance and sentiment scoring, and interaction with local LLM(Gemma3:1b), it offers users an efficient way to stay informed and query information related to defense and geopolitical events. The system is modular, allowing for potential expansion to include more sources or different analytical models in the future. Lastly, by utilizing local models running Ollama, it offers a degree of privacy and control over the analysis process.

Appendix 11: References

1. Business Intelligence and Analytics Software. Tableau. https://www.tableau.com/
2. Factbook. Factbook/factbook.json: World Factbook country profiles in JSON - free open public GitHub. https://github.com/factbook/factbook.json/tree/master
3. KirboCoder. Kirbocoder/Lockheed-Group-Project: Florida Polytechnic Lockheed Group Project. GitHub. https://github.com/KirboCoder/Lockheed-Group-Project/tree/main
4. Real Time World Statistics. Worldometer. https://www.worldometers.info/
5. WorldBank. World Bank Group - International Development, poverty and Sustainability. https://www.worldbank.org/ext/en/home
6. Pycountry. PyPI. https://pypi.org/project/pycountry/
7. Beautiful Soup Documentation. https://www.crummy.com/software/BeautifulSoup/bs4/doc/
8. Requests: HTTP for Humans™. https://requests.readthedocs.io/en/latest/
9. Hugging Face Transformers. https://huggingface.co/docs/transformers/index
10. NLTK : Natural Language Toolkit. https://www.nltk.org/
11. Scikit-learn: Machine Learning in Python. https://scikit-learn.org/stable/
12. FastAPI. https://fastapi.tiangolo.com/
13. Ollama. https://ollama.com/
14. Global Firepower. https://www.globalfirepower.com/
15. https://ourworldindata.org/age-structure
16. https://www.cia.gov/the-world-factbook/countries/germany/#people-and-society
17. https://data.worldbank.org/indicator/NY.GDP.PCAP.CD?locations=DE
18. https://ourworldindata.org/life-expectancy
19. https://www.globalexchange.es/monedas-del-mundo/euro
20. https://www.destatis.de/EN/Themes/Labour/Labour-Market/Unemployment/Tables/press-month2.html
21. https://www.statista.com/statistics/295519/germany-share-of-economic-sectors-in-gross-domestic-product/

22. https://data.worldbank.org/indicator/MS.MIL.XPND.GD.ZS?locations=DE

23. https://www.imf.org/external/datamapper/NGDP_RPCH@WEO/WEOWORLD/DEU

24. https://tradingeconomics.com/germany/unemployment-rate#:~:text=German%20Jobless%20Rate%20Unchanged%20at,below%20market%20forecasts%20of%205.9%25